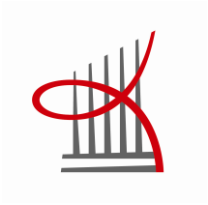


LANGATON TIEDONSIIRTO SULAUTETUSSA JÄRJESTELMÄSSÄ

Ville Rosila

Opinnäytetyö
Huhtikuu 2014
Tietotekniikan koulutusohjelma
Sulautetut järjestelmät ja elektroniikka

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Sulautetut järjestelmät ja elektroniikka

ROSILA, VILLE:
Langaton tiedonsiirto sulautetussa järjestelmässä

Opinnäytetyö 53 sivua, joista liitteitä 14 sivua
Huhtikuu 2014

Tässä opinnäytetyöraportissa käsitellään langattoman tiedonsiirron toteuttamista erääseen sulautettuun järjestelmään. Työn tärkeimpänä tavoitteena oli saada langaton tiedonsiirto toimimaan järjestelmässä tehokkaasti ja luotettavasti. Toisena tavoitteena oli toteuttaa jokin käytännönläheinen esimerkkisovellus, jossa langattomasta tiedonsiirrosta on hyötyä. Raportissa perehdytään työssä käytettyihin komponentteihin ja niiden oleellisimpiin teknisiin ominaisuuksiin. Komponenttien lisäksi esitellään langattoman tiedonsiirron ja sitä hyödyntävän esimerkkisovelluksen ohjelmalliset toteutukset.

Työssä toteutettu järjestelmä koostuu kahdesta kytkennöiltään identtisestä ohjainkortista, joiden välille langaton tiedonsiirto toteutettiin. Suurin yksittäinen komponentti molemmilla ohjainkorteilla on Olimexin valmistama AVR-MT-128-kehitysalusta, jonka keskeisin komponentti on Atmel AVR ATMega128 -mikrokontrolleri. Mikrokontrolleri ohjelmoitiin AVR Studio 4 -kehitysympäristössä C-kielellä. Kehitysalustan sisältämät komponentit tarjoavat paljon mahdollisuuksia erilaisten toimintojen toteuttamiseen ohjelmallisesti.

Varsinainen langaton tiedonsiirto järjestelmän ohjainkorttien välillä toimii Nordic Semiconductor nRF24L01+ -lähetin-vastaanotinpiireillä. Tiedonsiirrossa käytetään 2,4 GHz:n ISM-taajuusaluetta, joka ei edellytä erillistä lupaa. Lähetin-vastaanotinpiiriä ohjataan SPI-väylän kautta. Tässä työssä piiriä ohjaavana mikrokontrollerina on kehitysalustan ATMega128. Lähetin-vastaanotinpiiri sisältää ohjelmoitavia rekistereitä, joiden alustuksilla sille määritellään asetukset piiriä ohjaavan mikrokontrollerin kautta.

Ohjainkorttien välinen langaton tiedonsiirto saatiin toimimaan luotettavasti, joten työn tavoitteet saavutettiin. Esimerkkisovelluksena toteutettiin NTC-termistoriin perustuva lämpötila-anturi, jonka mittaamaa lämpötilatietoa pystytään siirtämään langattomasti järjestelmän ohjainkorttien välillä.

Asiasanat: sulautettu järjestelmä, langaton tiedonsiirto, nRF24L01+-lähetin-vastaanotinpiiri, AVR, SPI-väylä.

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in ICT Engineering
Embedded Systems and Electronics

ROSILA, VILLE:
Wireless Data Transmission in Embedded System

Bachelor's thesis 53 pages, appendices 14 pages
April 2014

This thesis proposes an embedded system, which includes wireless communication between two transceivers. The main goal was to get the wireless communication to work effectively and reliably. Another goal was to implement practical example application, which takes advantage of the wireless communication. This thesis introduces the required components and their essential technical specifications. The implementation of the wireless communication and example applications are covered as well.

The embedded system consists of two identical main boards, between which the wireless data transmission was implemented. The single biggest part of these boards is the Olimex AVR-MT-128 development board, which contains Atmel AVR ATMega128 microcontroller. Programming of the ATMega128 microcontroller was done in C language via AVR Studio 4 programming environment. The variety of components on the development board offers a great number of programmable features.

The wireless communication between the main boards was implemented via the Nordic Semiconductor nRF24L01+ transceiver. The transceiver uses 2.4 GHz ISM frequency band for the wireless communication, since the particular frequency band does not require permission to be used. Programming of the transceiver chip is done via the ATMega128 microcontroller. The transceiver and ATMega128 both support the SPI bus, which is used for the communication between these two components. The parameters for the transceiver are defined via a variety of programmable registries.

The wireless communication between the two transceivers was successfully implemented, so the goals that were set can be considered achieved. The practical example application consists of a temperature sensor, which was implemented by using an NTC thermistor for the measurement of the temperature. The temperature data can be wirelessly transmitted between the transceivers of the system.

Key words: embedded system, wireless data transmission, nRF24L01+ transceiver, AVR, SPI bus.

SISÄLLYS

ERITYISSANASTO

SPI	Serial Peripheral Interface. Tiedonsiirtostandardi, joka on toimintaperiaatteeltaan synkronoitu sarjadataväylä.
ISM	Industrial, Scientific and Medical radio band. Radiotaajuuskaista, jonka käyttö ei edellytä erillistä lupaa.
NTC	Negative Temperature Coefficient Thermistor. Termistori, jonka resistanssi pienenee lämpötilan kasvaessa.
LCD	Liquid Crystal Display. Nestekidenäyttö.
LED	Light-Emitting Diode. Puolijohdekomponentti, joka säteilee tietyn väristä valoa, kun sen läpi kulkee sähkövirta.
JTAG	Joint Test Action Group. Debuggausliittymä.
USB	Universal Serial Bus. Yleinen sarjaväyläarkkitehtuuri, jonka avulla liitetään mm. tietokoneisiin oheislaitteita.
ADC	Analog to Digital Converter. A/D-muunnin on elektroniikan laite, joka muuntaa jatkuvan analogisen signaalin digitaalisiksi lukuarvoiksi.
I/O	Input/Output. I/O-portit mahdollistavat tiedon lähettämisen laitteesta ulos tai vastaavasti tiedon vastaanottamisen.
AVR	Atmelin kehittämä mikrokontrolleriperhe, johon kuuluu 8-bittisiä mikrokontrollereita.
RISC	Reduced Instruction Set Computing. Suoritinarkkitehtuuri, jossa suorittimen konekieliset käskyt pyritään pitämään mahdollisimman yksinkertaisina ja nopeina suorittaa.
Flash-muisti	Yleinen ohjelmamuistityyppi, jossa tieto säilyy vaikka käyttöjännitteet katkaistaan.
SRAM	Static Random Access Memory. Muistityyppi, joka säilyttää tilansa, kunnes piirin käyttöjännitteet katkaistaan.
EEPROM	Electronically Erasable Programmable Read-Only Memory. Muistityyppi, joka säilyttää tilansa vaikka käyttöjännitteet katkaistaan.
MOSI	Master Out Slave In. SPI-väylän datajohdin, jota pitkin isäntälaite lähettää dataa orjalaitteelle.

MISO	Master In Slave Out. SPI-väylän datajohdin, jota pitkin orjalaite lähettää dataa isäntälaitteelle.
CLK	Clock. SPI-väylän johdin, jota pitkin kellosignaali viedään kaikille väylän laitteille synkronointia varten.
SS	Slave Select. SPI-väylän johdin, jonka avulla valitaan väylällä aktiivinen orjalaite.
GFSK	Gaussian Frequency-Shift Keying. Taajuusmodulaatiomenetelmä, jossa digitaalisen hyötysignaalin 1- ja 0-bitit muuttelevat kantoaallon taajuutta korkeammaksi tai matalammaksi.

1 JOHDANTO

Tämän opinnäytetyön tarkoituksena oli toteuttaa sulautetussa järjestelmässä toimiva langaton tiedonsiirto. Aiheena langaton tiedonsiirto on melko ajankohtainen, koska sen käyttö elektroniikan pienemmissäkin sovelluksissa yleistyy kovaa vauhtia. Työssä toteutettiin järjestelmä, joka koostuu kahdesta rakenteeltaan identtisestä ohjainkortista. Työn tavoitteena oli saada langaton tiedonsiirto toimimaan tehokkaasti ja luotettavasti molempiin suuntiin ohjainkorttien välillä.

Yksittäinen ohjainkortti koostuu useista komponenteista. Tärkein ja samalla suurin yksittäinen osa on Olimexin valmistama AVR-MT-128-kehitysalusta, jonka keskeisin komponentti työn kannalta on Atmelin valmistama ATMega128-mikrokontrolleri. Kehitysalustaan kytkettiin erillinen Nordic Semiconductorin valmistama nRF24L01+-lähetin-vastaanotinpiiri, jonka kautta langaton tiedonsiirto toimii 2,4 GHz:n ISM-taajuusalueella. Lähetinvastaanotinpiiriä ohjataan kehitysalustalla olevan mikrokontrollerin avulla SPI-väylän kautta. Mikrokontrolleri ohjelmoitiin AVR Studio 4 -kehitysympäristössä C-kielellä. Työn koko järjestelmä siis koostuu kahdesta tällaisesta ohjainkortista.

Tämän raportin alussa esitellään työssä käytettyjä laitteita sekä kerrotaan niiden oleellisimmat tekniset ominaisuudet. Raportissa perehdytään tarkemmin työssä käytetyn lähetinvastaanotinpiirin toimintaan sekä sen ohjaukseen ja ohjelmointiin ATMega128-mikrokontrollerin avulla. Järjestelmällä tehtiin NTC-termistorilla lämpötilaa mittaava esimerkkisovellus, jossa lämpötilatieto saadaan siirrettyä langattomasti järjestelmän sisällä.

2 TYÖSSÄ KÄYTETTY LAITTEISTO

Tässä luvussa esitellään tarkemmin laitteet, joita työssä käytettiin. Laitteiden oleellisimpiin teknisiin ominaisuuksiin perehdytään, koska työn onnistuminen edellyttää niiden tuntemusta. Opinnäytetyöraportissa pyritään kuitenkin keskittymään lähinnä työn kannalta oleellisiin teknisiin ominaisuuksiin. Lisää laitteiden teknisiä ominaisuuksia on saatavilla laitevalmistajien kotisivujen kautta ladattavissa olevista datalehdistä. Laitevalmistajat tarjoavat usein myös joitain ohjelmakoodiesimerkkejä, joita tässäkin työssä osittain hyödynnettiin.

2.1 Olimex AVR-MT-128 [1]

Yksi työn keskeisimpiä laitteita on Olimexin valmistama AVR-MT-128-kehitysalusta (kuva 1). Kehitysalustan koko kytkentäkaavio on tämän raportin lopussa liitteessä 1.

KUVA 1. Olimex AVR-MT-128-kehitysalusta edestä kuvattuna [1]

Kehitysalusta on monipuolinen kokonaisuus, joka koostuu suuresta määrästä erilaisia toimintoja mahdollistavia komponentteja. Komponentit on koottu samalle piirilevylle koko kehitysalustaa ohjaavan mikrokontrollerin kanssa. Kehitysalustalla olevan mikrokontrollerin ohjelmointi tapahtuu PC:n USB-liitynnän kautta. AVR Studio 4 -ohjelmointiympäristössä kehitetty ja käännetty ohjelma kirjoitetaan mikrokontrollerin flash-muistiin JTAG-liitynnästä. JTAG-liitin kytkettiin työtä tehtäessä Olimexin valmistamaan AVR-JTAG-USB-emulaattoriin, joka muuntaa PC:n USB-portista tulevan datan

kehitysalustan JTAG-standardille sopivaan muotoon. Kyseisen kehitysalustan mikrokontrollerina toimii ATMega128.

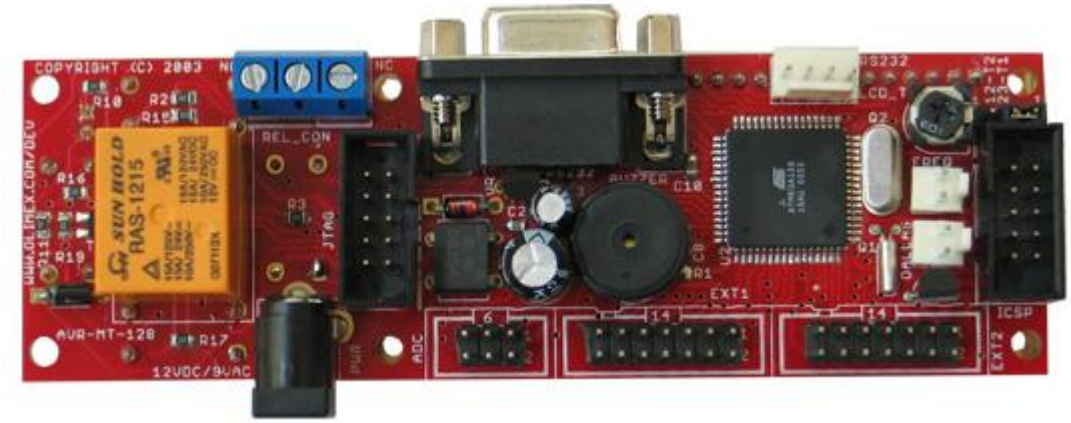
2.1.1 Kehitysalustan komponentit ja ominaisuudet

Kehitysalustalla on mikrokontrollerin lisäksi paljon muitakin komponentteja, jotka mahdollistavat erilaisia ohjelmallisesti toteutettavia toiminnallisuuksia. Tällaisia komponentteja ovat esimerkiksi kuvassa 1 näkyvät kytkimet (5 kappaletta), joiden avulla on mahdollista toteuttaa käyttöliittymä. Kehitysalustalla on myös rele, jonka rinnalle on kytketty punainen LED. Releen käyttö osoittautui hyväksi tavaksi todeta ohjelmakoodin toiminta työtä tehtäessä. Kytkinten vieressä on 2x16-merkkinen LCD-näyttö. Näytölle siis mahtuu kerrallaan 32 kappaletta ASCII-merkkejä.

Kehitysalustan käyttöjännite on 12 V, mutta monet alustan komponenteista kuitenkin tarvitsevat vain 5 V:n jännitteen toimiakseen. Tämä on alustalla toteutettu 78L05-regulaattorilla, joka muuntaa 12 V suuruisen jännitetason 5 V:n suuruiseksi.

2.1.2 Kehitysalustan liittimet

Kehitysalustan kääntöpuolella on käytännöllisiä piikkirimoja, jotka on kytketty mikrokontrollerin pinneihin (kuva 2).



KUVA 2. Olimex AVR-MT-128-kehitysalusta kääntöpuolelta kuvattuna [1]

Kehitysalustan liittimistä suuri osa on kytketty työn kannalta oleellisiin mikrokontrollerin pinneihin. Alustan kytkentäkaaviossa (liite 1) tämän työn kannalta tärkeimmät liittimet on merkitty seuraavilla nimillä: EXT1, EXT2, ADC ja JTAG. JTAG-liitin on oleellinen osa, koska ohjelmakoodi kirjoitetaan mikrokontrollerin flash-muistiin sitä kautta.

EXT1- ja EXT2-liittimiin on kytketty mikrokontrollerin I/O-porttien pinnejä. Niihin lukeutuu muun muassa mikrokontrollerin SPI-väylä, jota tässä työssä hyödynnettiin. ADC-liitin kattaa lähes kaikki mikrokontrollerin A/D-muunnosmahdollisuudet, joita hyödynnettiin työn esimerkkisovelluksessa. Työn laitteiston kytkennässä hyödynnetyt liittimet sekä pinnit esitellään tarkemmin ohjelmoinnin yhteydessä.

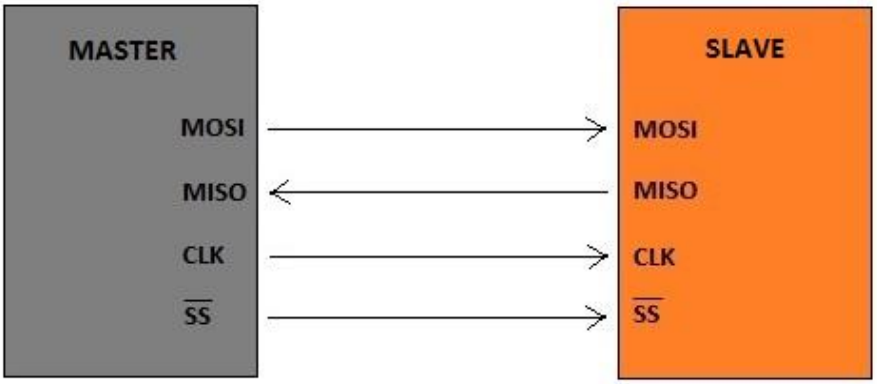
2.2 AVR ATMega128 -mikrokontrolleri [2]

ATMega128 on Atmelin suunnittelema mikrokontrolleri, joka kuuluu 8-bittiseen AVR-mikrokontrolleriperheeseen. Kyseisen mikrokontrolleriperheen kaikkien tuotteiden ydin pohjautuu RISC-arkkitehtuuriin. ATMega128 on pakattu 64-jalkaiseen koteloon ja siihen sisältyy 128 kB (128 kilotavua) ohjelmoitavaa flash-muistia, 4 kB SRAM-työmuistia rekistereitä ja muita muuttujia varten sekä 4 kB EEPROM-muistia. Mikrokontrolleriin sisältyy myös 8-kanavainen A/D-muunnin, jonka resoluutio on 10 bittiä. A/D-muunninta ja sen käyttöä käsitellään tarkemmin esimerkkisovelluksen yhteydessä.

Kehitysalustalla 5 V käyttöjännite tulee mikrokontrollerille suoraan regulaattoripiiriltä. Suurin mahdollinen kellotaajuus on 16 MHz, jonka piiri saa kehitysalustalla sijaitsevalta 16 MHz:n kiteeltä. Vapaasti ohjelmoitavia I/O-pinnejä kontrollerissa on 53 kappaletta.

2.2.1 SPI-väylä

Yksi ATMega128:n tärkeimpiä teknisiä ominaisuuksia opinnäytetyön kannalta on SPI-väylä. Työssä käytetty radiomoduuli nRF24L01+ käyttää SPI-väylää tiedonsiirtoon. SPI-väylä koostuu vähintään neljästä johtimesta, jotka ovat MOSI, MISO, CLK ja SS.



KUVA 3. SPI-väylän toimintaa havainnollistava kaavio

Kuvan 3 tapauksessa SPI-väylän isäntälaitteena (MASTER) toimii ATMega128, kun taas orjalaitteena (SLAVE) toimii nRF24L01+-lähetin-vastaanotin. MOSI-johdinta (Master Out Slave In) pitkin isäntälaite lähettää orjalaitteelle dataa. Vastaavasti MISO-johdinta (Master In Slave Out) pitkin orjalaite lähettää isäntälaitteelle dataa. SPI-väylä onkin niin sanottu kaksisuuntainen (Full Duplex) tiedonsiirtostandardi, koska dataa voidaan lähettää molempiin suuntiin samanaikaisesti.

SPI-väylän tiedonsiirto on synkronoitua, mikä tarkoittaa sitä, että datan luku ja kirjoitus tahdistetaan laitteiden välillä yhteiseen kellosignaaliin. Sovelluskytkennässä datan luku ja kirjoitus dataväylillä tapahtuu aina kellosignaalin nousureunan hetkellä. SPI-väylän kellosignaalin määrää isäntälaite (ATMega128), jolle kellosignaalin taajuus määritetään ohjelmallisesti. SS-signaalijohdinta käytetään väylällä aktiivisen orjalaitteen valitsemiseen. Tässä tapauksessa orjalaitteita on vain yksi, mutta jos orjalaitteita olisi useampia, tarvittaisiin jokaiselle orjalaitteelle oma SS-signaalijohdin. Kuvassa 3 SS-signaalin päälle merkitty viiva kuvastaa sitä, että signaali on 0-tilassa aktiivinen. SS-signaali on siis lepotilassa aina 1-tilassa, mutta asetetaan 0-tilaan orjalaitteen valitsemiseksi.

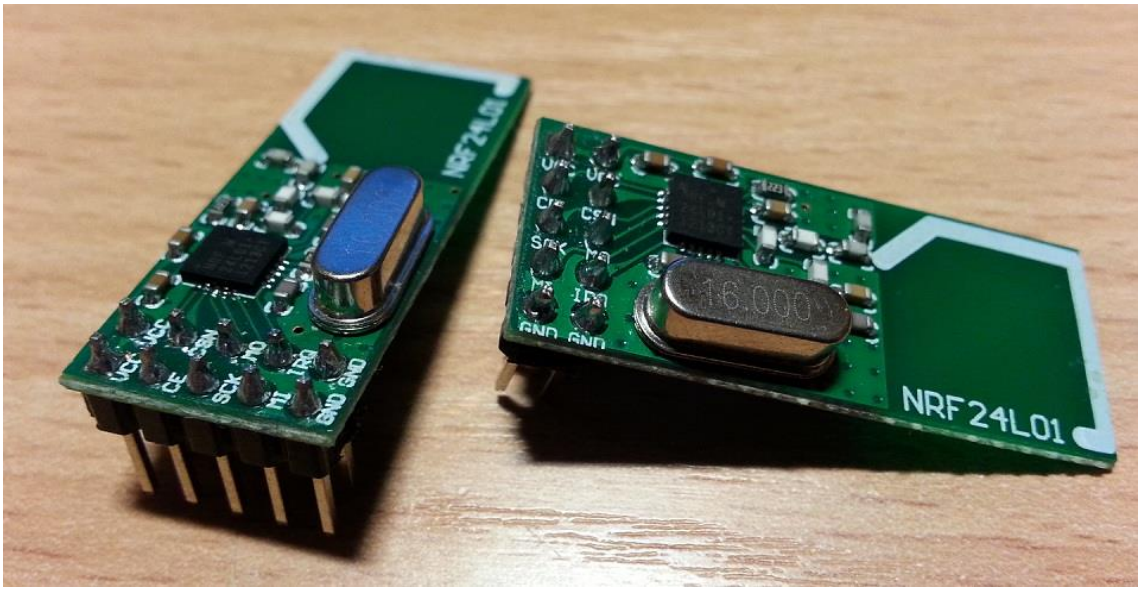
2.2.2 Keskeytyspalvelut

Eräs ATmega128-mikrokontrollerin ominaisuus on keskeytyspalvelut. Mikrokontrollerissa voidaan käyttää sekä ulkoisista lähteistä tulevia keskeytyksiä, että mikrokontrollerin sisäisiä keskeytyksiä. Sisäisiä keskeytyksiä ovat esimerkiksi ajastinohjatut keskeytykset tai vaikkapa A/D-muunnoksen aiheuttama keskeytys.

Tässä opinnäytetyössä hyödynnettiin useita eri keskeytystyyppejä. Oleellisimpana on nRF24L01+-lähetin-vastaanotinpiiriltä tuleva ulkoinen keskeytyspyyntö. Tämä keskeytyspyyntö tapahtuu aina, kun radiopiirin lähetys- tai vastaanottorekisteriin saapuu dataa. Toinen tärkeä keskeytystyyppi on mikrokontrollerin sisäisellä ajastimella toimiva keskeytys, jota hyödynnettiin kehitysalustan kytkintilojen lukemisessa. Myös A/D-muunnoksen valmistumisen aiheuttamaa keskeytystä hyödynnettiin esimerkkisovelluksessa. Kaikkien keskeytystyyppien toimintaan ja niiden käyttöön perehdytään tarkemmin ohjelmoinnin ja esimerkkisovelluksen yhteydessä.

2.3 nRF24L01+-lähetin-vastaanotin [3]

Nordic Semiconductorin valmistama mikropiiri nRF24L01+ on langattomaan tiedonsiirtoon tarkoitettu lähetin-vastaanotin. Se on käytännöllinen erityisesti sulautetuissa järjestelmissä, koska sen virrankulutus on pyritty pitämään matalana. Piirin langaton tiedonsiirto tapahtuu 2,4 GHz:n ISM-taajuusalueella (2,4000 - 2,4835 GHz), ja kyseisen taajuusalueen käyttö ei edellytä erillistä lupaa. Opinnäytetyössä käytettiin radiomoduuleita, joissa oli nRF24L01+-lähetin-vastaanotinpiirin lisäksi valmiina kaikki sen käyttöön tarvittavat komponentit, kuten antenni ja 16 MHz:n kide (kuva 4).



KUVA 4. Työssä käytettyjä radiomoduuleita, joissa on nRF24L01+-lähetin-vastaanotinpiiri

Kuvassa 4 piirilevyllä näkyvä mikropiiri on nRF24L01+. Piirilevyn päädyssä oleva piikkirima kattaa tiedonsiirtoon sekä käyttöjännitteensyöttöön vaadittavat kahdeksan johdinta. Vastakkaisessa päässä piirilevyä näkyvä koukun muotoinen folio toimii lähetin-vastaanotinpiirin antennina.

2.3.1 nRF24L01+-piirin ominaisuudet

Valmistaja ilmoittaa datalehdessä piirin käyttöjännitteeksi 1,9 - 3,6 V. Tämä tulee ottaa huomioon laitteiden kytkentävaiheessa, koska työssä käytetty kehitysalusta sisältää vain 5 V:n tasoiset I/O-liitynnät. Piirin virrankulutus on matala, joten se soveltuu erinomaisesti esimerkiksi akkukäyttöisiin elektroniikan sovelluksiin (taulukko 1).

TAULUKKO 1. nRF24L01+ -piirin virrankulutus eri tilanteissa

Piirin tila	Lähetys suurimmalla teholla	Vastaanotto suurimmalla nopeudella	Valmiustilat I ja II	Horrostila
Piirin virrankulutus	11,3 mA	13,5 mA	26 µA - 320 µA	900 nA

Piiri käyttää langattomassa tiedonsiirrossa GFSK-modulaatiota, jonka avulla hyötysignaali sidotaan 2,4 GHz:n kantoaaltoon ennen radiotielle lähettämistä. Kuten aiemmin jo mainittiin, nRF24L01+-lähetin-vastaanotinpiirin ohjaus tapahtuu ulkoisella mikrokontrollerilla SPI-väylän kautta. Piirin SPI-väylän suurin tiedonsiirtonopeus on 10 Mbit/s.

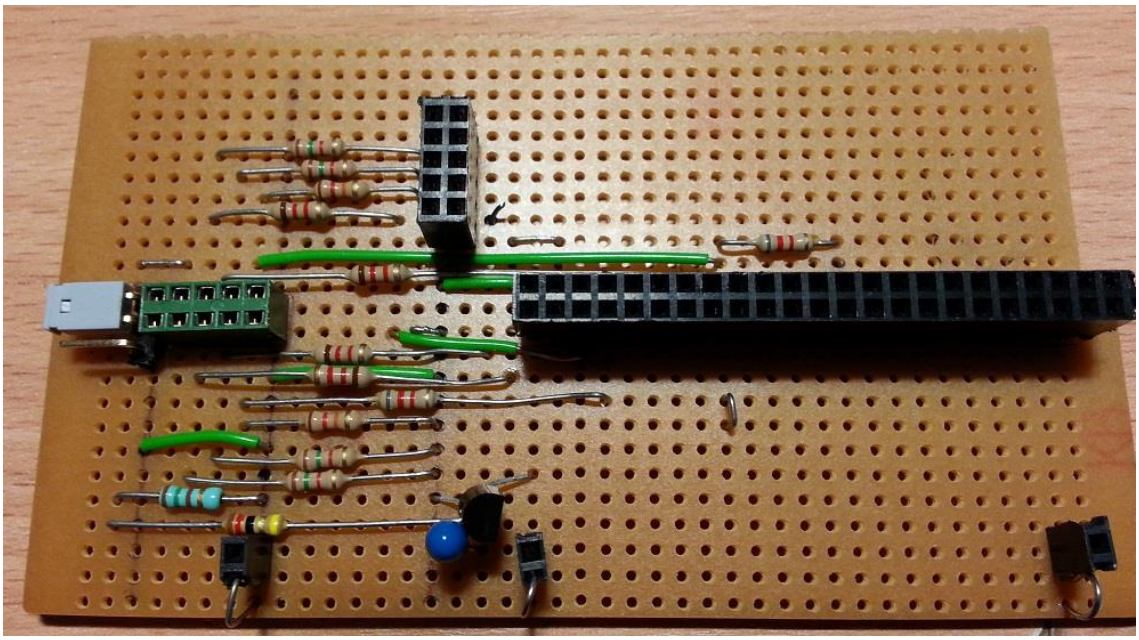
Varsinaista ohjausta varten piiri sisältää 29 kappaletta SPI-väylän kautta ohjelmoitavissa olevia rekistereitä (nRF24L01+-datalehden taulukko 28, s. 57). Näiden rekisterien avulla on mahdollista asettaa esimerkiksi lähetysteho (0 dBm, -6 dBm, -12 dBm tai -18 dBm) ja siirtonopeus (250 kbit/s, 1Mbit/s tai 2 Mbit/s) halutuksi. Piirin toimiessa vastaanottimena sen herkkyys on -94 dBm siirtonopeudella 250 kbit/s. Watteina ilmaistuna -94 dBm vastaa 0,4 pW suuruista tehoa.

2.3.2 Radiomoduulin liittimet

Lähetin-vastaanotinmikropiirissä on kytkentäjalkoja yhteensä kaksikymmentä kappaletta, mutta moduulin ja mikrokontrolleriosan välisessä liitynnässä tarvitaan vain kahdeksan johdinta. Johtimista kahta käytetään radiomoduulin käyttöjännitteen (V_{CC} ja GND) kytkemiseen ja loppuja tiedonsiirtoon lähetin-vastaanotinpiiriä ohjaavan mikrokontrollerin kanssa. Tiedonsiirtoon ja lähetin-vastaanotinpiirin ohjaamiseen tarvittaviin liittimiin lukeutuvat SPI-väylän vaatimat johtimet: MOSI, MISO, CLK ja SS. Näiden johtimien lisäksi piiri kuitenkin tarvitsee myös IRQ- ja CE-johtimet. IRQ-johtimen kautta mikrokontrolleri tulkitsee nRF24L01+-lähetin-vastaanottimen datan lähetyksen ja vastaanoton yhteydessä tapahtuvat keskeytyspyynnöt. CE-johdin (Chip Enabled) puolestaan määrää nRF24L01+-lähetin-vastaanotinpiiriä aloittamaan datan lähetyksen tai vastaanoton antennin kautta.

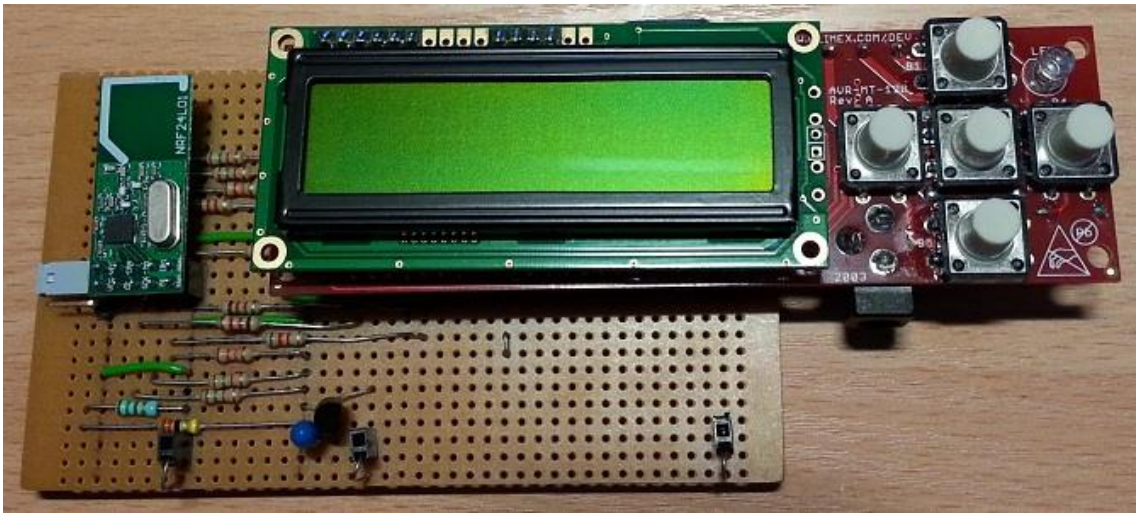
2.4 Laitteiden kytkentä

Olimexin mikrokontrollerikortti ja lähetin-vastaanotinmoduuli kytkettiin toisiinsa kuvan 5 mukaisella koekytkentälevyllä.



KUVA 5. Toinen työssä käytetyistä koekytkentälevyistä

Koekytkentälevyille on kuvassa 5 näkyvät liittimet kytketty siten, että kehitysalustan ja radiomoduulin liittimet sopivat niihin suoraan. Kytkentälevyillä ovat myös kaikki tarpeelliset komponentit lähetinvastaanotinpiirin edellyttämiä tasomuunnoksia varten. Kehitysalustan ja radiomoduulin kytkeminen toisiinsa koekytkentälevyn kautta tapahtuu asettamalla molemmat osat niille tarkoitettuihin liittimiin (kuva 6).



KUVA 6. Kaikki tarvittavat komponentit koekytkentälevyllä

Työn koko järjestelmä koostuu kahdesta kappaleesta kuvan 6 mukaisia kokonaisuuksia. Koska nRF24L01+-piirin käyttöjännitteen on oltava 1,9 – 3,3 V, ei sitä voi kytkeä suoraan kehitysalustan 5 V:n jännitteeseen. Koekytkentälevyillä käyttöjännite alennetaan regulaattorilla, joka muuntaa 5 V:n jännitetason 3,3 V:n tasolle.

2.5 Ohjelmointiympäristö AVR Studio 4

Ohjelmisto kehitettiin AVR Studio 4 -ohjelmointiympäristössä C-kielellä. Myös C++- ja assembly-kielen käyttö olisi ollut mahdollista. AVR Studio 4 mahdollistaa ohjelmointiympäristön tukemien mikrokontrollereiden toiminnan simuloinnin. Tämä ominaisuus osoittautui työn alkuvaiheessa käteväksi, kun ei vielä ollut kaikkia työhön tarvittavia fyysisiä osia saatavilla.

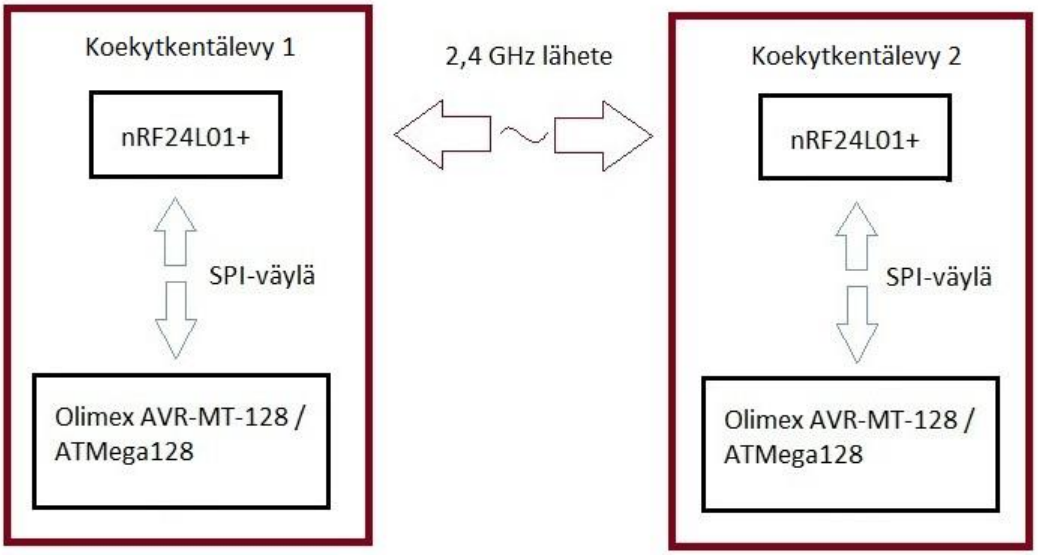
Ohjelmointiympäristössä on myös kattavat mahdollisuudet käytössä olevan mikrokontrollerin toiminnan seurantaan ohjelmakoodin ajon aikana. Debug-toiminto mahdollistaa JTAG-liitynnän kautta ohjelmakoodin ajamisen eteenpäin askel kerrallaan, mikä helpottaa mikrokontrollerin tilan ja toiminnan seurantaa. Tämä oli suuri apu työtä tehtäessä, koska välillä ongelmakohtien löytäminen koodista on työlästä.

3 TYÖN TOTEUTTAMINEN JA OHJELMOINTI

Tässä luvussa kerrotaan tarkemmin itse työn toteuttamisesta ja saaduista tuloksista. Pääpaino tässä luvussa on kuitenkin työn ohjelmallisen osuuden toteuttamisessa, koska se oli ehdottomasti suurin osa työtä.

3.1 Työn suunnittelu

Opinnäytetyön suunnittelu aloitettiin keväällä 2014 tammikuun ja helmikuun aikana, ennen kuin varsinaisia osia työn toteuttamiseksi saatiin. Työn suurimmaksi tavoitteeksi asetettiin saada langaton tiedonsiirto toimimaan kahden nRF24L01+-lähetin-vastaanotinpiirin välillä (kuva 7).



KUVA 7. Järjestelmän toimintaa kuvaava lohkokaavio

Myös työn ohjelmiston toteuttamista ja rakennetta suunniteltiin jo helmikuussa AVR Studio 4 -ohjelmointiympäristön simulaattorilla. Huomattava määrä aikaa käytettiin nRF24L01+-lähetin-vastaanotinpiirin datalehden tulkitsemiseen jo ennen varsinaisen ohjelmoinnin aloittamista, koska piiri oli entuudestaan melko tuntematon. Myös muutamaan internetistä löytyneeseen nRF24L01+-lähetin-vastaanotinpiiriä hyödyntäneeseen ohjelmistoon tutustuttiin. Varsinainen ohjelmointi aloitettiin maaliskuun alussa, kun tarvittavat osat saatiin käyttöön.

3.2 Ohjelmointi [4], [5]

Tässä kappaleessa esitellään ohjelmistoon vaadittavat alustukset ja eri toiminnallisuuksia suorittavat aliohjelmat, jotka ovat tarpeellisia langattoman yhteyden aikaansaamiseksi. Esitettävät ohjelmakoodit, joilla langaton tiedonsiirto saadaan toimimaan, koskevat molempia järjestelmän ohjainkortteja. Ohjainkorttien välille tulee huomattavia eroja vasta kappaleen 4 esimerkkisovelluksen myötä. Kaikki tarpeelliset ohjelmistot löytyvät raportin liitteistä (liitteet 2, 3 ja 4).

3.2.1 Mikrokontrollerin alustukset

Ensimmäinen askel työn ohjelmointivaiheessa oli toteuttaa kaikki ATMega128:n tarvitsemat alustukset työssä käytettävien ominaisuuksien osalta. Oleellista on alustaa porttikohtaiset suuntarekisterit, jotta mikrokontrollerin pinnit toimivat oikeaan suuntaan (input vai output). Ohjelmiston rakenteen selkeyttämiseksi kaikki alustukset lajiteltiin omiin aliohjelmiinsa, jotka ajetaan kerran mikrokontrollerin käynnistyksen yhteydessä. Alustukset aloitettiin tarvittavien I/O-porttien suuntarekistereistä (kuva 8).

```
/*****************************************************************************************/
// IO-porttien alustukset
void IO_Init(void)
{
    DDRA = 0x40;     // PA1...PA4/Kytkimet = in, PA6/Rele = out
    DDRG = 0x01;     // PG0/CE = out
}
```

KUVA 8. I/O-porttien suuntarekisterien alustukset

Kuvan 8 mukainen aliohjelma IO_Init suoritetaan pääohjelmassa kerran mikrokontrollerin käynnistyksen yhteydessä. IO_Init-aliohjelma alustaa portin A suuntarekisteriin DDRA (ATMega128 datalehti, s. 86) heksadesimaaliarvoksi 0x40, joka vastaa binäärilukua 0b01000000 (Raportissa 0x merkitsee heksadesimaalilukua, 0b binäärilukua). Portin A viisi ensimmäistä pinniä (PA0 - PA4) on kytketty kehitysalustan kytkimiin, joten ne alustetaan inputeiksi asettamalla niitä vastaavat bitit nollaan DDRA-rekisterissä. Suuntarekisterin kuudes bitti puolestaan vastaa portin A pinniä (PA6), johon kehitysalustalla sijaitseva rele on kytketty. Kyseinen bitti asetetaan arvoon 1, jolloin pinni alustetaan outputiksi. Samassa aliohjelmassa alustetaan portin G suuntarekisteriin DDRG (ATMega128 datalehti, s. 88) arvoksi 0x01, jolloin portin G nollas pinni (PG0)

asettuu outputiksi. Kyseinen pinni alustetaan outputiksi, koska se on kytketty radiomoduulin CE-pinniin.

Seuraavaksi pääohjelmamassa ajetaan LCD_Init-aliohjelma, jossa alustetaan kehitysalustan LCD-moduuli toimintavalmiiksi. LCD-moduulia ohjaavat aliohjelmat lainattiin työtä varten suoraan kehitysalustan tuotesivulta löytyvistä esimerkkiohjelmista [1]. Ohjelmakoodi on kokonaisuudessaan raportin liitteessä 3. LCD-moduulin jälkeen siirrytään SPI_Init-aliohjelmaan, jossa alustetaan SPI-väylän asetukset, sekä tarvittavat I/O-portit SPI-väylän datajohtimia varten (kuva 9).

```
/*****************************************************************************************/
// SPI-väylän alustukset
void SPI_Init(void)
{
    DDRB = 0x07;     // PB0/SS = out, PB1/CLK = out, PB2/MOSI = out, PB3/MISO = in
    SPCR = 0x52;     // SPI Enable=1, Master=1, SPR1=1 ja SPR0=0, jolloin
                     // fosc/64 = 250kHz eli 250 kbit/s siirtonopeus

    PORTB |= 0x01;  // SS high alustus
    PORTG &= ~0x01; // CE low alustus
}
```

KUVA 9. SPI-väylän edellyttämät alustukset

SPI-väylän datajohtimet SS, CLK, MOSI ja MISO sijaitsevat mikrokontrollerin portin B pinneissä PB0 - PB3. Kaikki muut johtimet paitsi MISO alustetaan outputeiksi datasuuntarekisterissä DDRB. SPCR-rekisteri määrittelee SPI-väylän asetukset (ATMega128 datalehti, s. 166). SPI-väylän siirtonopeus on sidoksissa mikrokontrollerin kideoskillaattoriin siten, että SPCR-rekisterin kaksi vähiten merkitsevää bittiä (SPR1 ja SPR0) määräävät esijakajan kideoskillaattorin taajuudelle (ATMega128 datalehden taulukko 72, s. 167). Työssä käytetyillä alustuksilla (SPR1=1 ja SPR0=0) väylän siirtonopeudeksi tulee 250 kbit/s.

$$\frac{f_{osc}}{64} = \frac{16\ \text{MHz}}{64} = 250\ \frac{\text{kbit}}{\text{s}} \qquad (3.1)$$

SPCR-rekisterissä asetetaan myös SPI-väylä aktiiviseksi (SPE=1), sekä ATMega128 kyseisen väylän isäntälaitteeksi (MSTR=1). Lopuksi aliohjelmassa asetetaan SPI-väylän SS-signaali 1-tilaan sekä radiomoduulia kontrolloiva CE-signaali 0-tilaan. Seuraavaksi pääohjelmassa ajetaan tarpeellisia keskeytyksiä varten tehtävät alustukset Interrupt_Init-aliohjelmassa (kuva 10).

```
/*****************************************************************************/
// Keskeytysten alustus
void Interrupt_Init(void)
{
    DDRE &= ~0x40;      // PE6/INT6 = in (IRQ)
    EICRB = 0x20;       // INT6 laskeva reuna aiheuttaa keskeytyspyynnön (bitit 5:4 = 10)
    EIMSK = 0x40;       // INT6 -keskeytyksen sallinta

    KesHetki = 20000;       // Ensimmäisen OCR1A-keskeytyksen ajanhetki 10 ms resetistä
    OCR1A = KesHetki;
    TCCR1B = 0x02;          // 0000 0010 clk/8
    TIMSK = 0x10;           // 0001 0000
}
```

KUVA 10. Tarpeellisten keskeytysten alustukset

Mikrokontrollerin portin E kuudes pinni (PE6) on kytketty radiomoduulin IRQ-johtimeen, joten se alustetaan inputiksi ulkoisten keskeytyspyyntöjen vastaanottoa varten. EICRB-rekisterillä määritellään ulkoisiin keskeytyspyyntöihin liittyvät kriteerit (ATMega128 datalehden taulukko 50, s. 90). Tässä tapauksessa keskeytyksen halutaan tapahtuvan tarkkailtavan signaalin laskevalla reunalla, koska nRF24L01+-lähetin-vastaanotinpiiriltä tuleva IRQ on alhaalla aktiivinen. Ulkoiset keskeytykset täytyy myös erikseen sallia EIMSK-rekisterissä (ATmega128 datalehti, s. 91).

Ulkoisten keskeytysten lisäksi työssä hyödynnettiin mikrokontrollerin ajastimiin pohjautuvia keskeytyksiä kehitysalustan kytkintilojen lukemiseen. TCCR1B-rekisterissä alustetaan Timer/Counter1-laskurin kellotaajuuden määrittävä esijakaja kolmella vähiten merkitsevällä bitillä (ATMega128 datalehden taulukko 62, s. 136). Kellotaajuus lasketaan kideoskillaattorin 16 MHz:n taajuudesta, aivan kuten SPI-väylän siirtonopeuden yhteydessä. Tässä tapauksessa etujakajan arvoksi alustetaan 8 (CS12=0, CS11=1, CS10=0), jolloin laskurin kellotaajuudeksi tulee 2 MHz. Tämän jälkeen Timer/Counter1-vertailukeskeytyspalvelu täytyy vielä sallia TIMSK-rekisterissä (ATMega128 datalehti, s. 128). OCR1A-vertailurekisteriin tallennetaan arvo, johon mikrokontrollerin laskurissa juoksevaa arvoa verrataan. Ensimmäisen OCR1A-keskeytyksen halutaan tapahtuvan 10 ms:n kuluttua mikrokontrollerin käynnistyksestä. Rekisterin arvoksi asetetaan 20000, koska esijakajalla asetettu kellotaajuus on 2 MHz. Kyseisellä kellotaajuudella 20000 kellojaksoa kestää tasan 10 ms, jolloin haluttuun aikaan päästään. Varsinaiset keskeytyspalveluissa tehtävät toiminnallisuudet esitellään myöhemmin, koska ne liittyvät oleellisesti nRF24L01+-lähetin-vastaanotinpiirin ohjaukseen.

3.2.2 SPI-väylän käyttö

Toinen vaihe ohjelmoinnissa oli luoda tarvittavat aliohjelmat SPI-väylän tiedonsiirtoa varten. Aliohjelmia luotiin kaksi kappaletta, joista ensimmäinen lähettää dataa tavu kerrallaan mikrokontrollerilta nRF24L01+-piirille (kuva 11).

```
/*************************************************************************************************/
// Yksittäisen tavun lähetys SPI-väylällä
void SPI_MasterTransmit(unsigned char cData)
{
    SPDR = cData;                // Tavu SPI-väylän datarekisteriin
    while(!(SPSR & (1<<SPIF))); // Odotetaan, että lähetys loppuu
}
```

KUVA 11. Yksittäisen tavun lähetys SPI-väylälle

SPI_MasterTransmit-aliohjelma on tehty datalehden esimerkkiohjelman perusteella (ATMega128 datalehti, s. 164). Aliohjelma saa parametrina yhden tavun, jonka se kirjoittaa SPI-väylän datarekisteriin lähetystä varten. Tämän jälkeen ohjelma odottaa while-silmukassa, kunnes SPIF-lippu kääntyy SPSR-rekisterissä lähetyksen valmistumisen merkiksi. Toinen SPI-väylän tiedonsiirtoa varten tehty aliohjelma vastaanottaa nRF24L01+-piiriltä paketin datan lähettämisen lisäksi (kuva 12).

```
/*************************************************************************************************/
// Yksittäisen tavun lähetys ja luku SPI-väylällä
char SPI_MasterTransmitRead(unsigned char cData)
{
    SPDR = cData;                // Tavu SPI-väylän datarekisteriin
    while(!(SPSR & (1<<SPIF))); // Odotetaan, että lähetys loppuu
    return SPDR;                 // Paluuarvona vastaanotettu data
}
```

KUVA 12. Yksittäisen tavun lähetys ja luku

SPI_MasterTransmitRead-aliohjelman ainut ero kuvan 11 aliohjelmaan on se, että aliohjelma saa paluuarvokseen SPI-väylän datarekisteriin mahdollisesti vastaanotetun tavun. Datarekisteriin vastaanotettu tavu voi sisältää esimerkiksi nRF24L01+-piirin langattomasti vastaanottaman paketin sisältämää dataa. Nämä kaksi aliohjelmaa periaatteessa riittävät laitteiden väliseen kommunikaatioon SPI-väylällä. Lisäksi kuitenkin tarvitaan vielä yksi aliohjelma, jonka avulla näitä tiedonsiirtoon tehtyjä aliohjelmia käytetään (kuva 13).

```
/*****************************************************************************************/
// Kommunikaatio nRF24L01+ -piirin kanssa SPI-väylän avulla
uint8_t *ReadWriteNRF(uint8_t ReadWrite, uint8_t NRFreg, uint8_t *package, uint8_t amount)
{
    static uint8_t ret[bytes];

    // Jos kirjoituskomento niin lisätään kyseisen rekisterin osoite komentotavuun
    if(ReadWrite == W)
    {
        NRFreg = W_REGISTER + NRFreg;
    }

    _delay_us(10);
    PORTB &= ~0x01;          // SS low, slave-laite aktiiviseksi
    _delay_us(10);
    SPI_MasterTransmit(NRFreg);     // Luku/kirjoituskomennon lähetys piirille
    _delay_us(10);

    int i;
    for(i=0; i<amount; i++)
    {
        if(ReadWrite == R && NRFreg != W_TX_PAYLOAD)    // Datan luku piiriltä
        {
            ret[i]=SPI_MasterTransmitRead(NOP);
            _delay_us(10);
        }
        else        // Datan lähetys piirille
        {
            SPI_MasterTransmit(package[i]);
            _delay_us(10);
        }
    }

    PORTB |= 0x01;       // SS high, slave-laite lopettaa vastaanoton
    return ret;          // Paluuarvona piiriltä luettu paketti, jos sellainen tuli
}
```

KUVA 13. Laitteiden välistä kommunikaatiota SPI-väylällä hallinnoiva aliohjelma

ReadWriteNRF-aliohjelma hallinnoi kaikkea liikennettä mikrokontrollerin ja lähetin-vastaanotinpiirin välillä. Parametreina aliohjelmalle tulee neljä muuttujaa, joiden perusteella aliohjelmassa toimitaan. Ensimmäinen parametri (ReadWrite) määrittää, suoritetaanko datan lähetys- vai lukutoiminto. Toinen parametri (NRFreg) sisältää sen rekisterin osoitetiedon, jota toiminto koskee. Kolmas parametri (*package) on osoitinmuuttuja, joka sisältää tiedon muistiosoitteesta, jossa lähetin-vastaanotinpiirille lähetettävä paketti sijaitsee. Viimeisenä parametrina (amount) aliohjelmalle tulee tieto SPI-väylällä lähetettävien tai luettavien tavujen määrästä.

Aliohjelman alussa luodaan taulukko, jonka solumääräksi tulee ohjelmiston alussa määritetty radiotiellä liikkuvien pakettien tavumäärä (3 tavua). Seuraavaksi tutkitaan ehtolauseella, halutaanko suorittaa kirjoituskomento. Kirjoituskomennon tapauksessa kirjoituskomentotavuun lisätään kyseessä olevan rekisterin osoite. Tämän jälkeen SPI-väylän SS-signaali käännetään 0-tilaan, jolloin lähetin-vastaanotinpiiri valmistautuu vastaanottamaan dataa SPI-väylältä. Ensin väylälle lähetetään kyseessä oleva komento, sitten ohjelmassa tulee for-rakenne, jossa joko lähetetään tai vastaanotetaan dataa kolmen tavun verran. Aliohjelman lopussa SPI-väylän SS-signaali käännetään takaisin 1-tilaan, jolloin nRF24L01+-piiri lopettaa datan vastaanoton väylältä. Paluuarvona aliohjelmalla on SPI-väylältä mahdollisesti luettu 3-tavuinen datapaketti.

3.2.3 nRF24L01+-lähetin-vastaanotinpiirin alustukset

SPI-väylän avulla voidaan aloittaa nRF24L01+-lähetin-vastaanotinpiirin ohjelmoitavien rekisterien alustus. Piirin ohjaamiseen tarvittavat komentotavut löytyvät valmistajan datalehdestä (taulukko 20, s. 51). Myös ohjelmoitavien rekisterien osoitteet ja merkitykset selityksineen löytyvät valmistajan datalehdestä (taulukko 28, s. 57–63). Tässä osuudessa työtä hyödynnettiin internetistä ladattavissa olevaa kirjastoa, jossa oli määritelty piirin rekisterien osoitteet valmiiksi niiden nimen mukaan. Kirjasto oli kuitenkin tehty vanhempaa nRF24L01-piiriä varten, ja siinä oli muutamia puutteita +-malliin verrattuna, joten kirjastoon täytyi tehdä lisäyksiä (liite 2).

Tarvittavat alustukset suorittava nRF24L01p_Init-aliohjelma ajetaan mikrokontrollerin käynnistyksen yhteydessä kerran. Alustukset suoritetaan SPI-väylän tiedonsiirtoa suorittavia aliohjelmia kutsumalla (kuva 14).

```
/*****************************************************************************************/
// nRF24L01+ alustus
void nRF24L01p_Init(void)
{
    _delay_ms(1.5); // Pieni delay, jotta piiri on varmasti standby-tilassa
    uint8_t package[5];

    // Sallitaan automaattiset vastaukset vastaanottavalta piiriltä (auto-acknowledgements)
    // Tämä mahdollistaa automaattiset lähetysten uudelleenyritykset
    package[0]=0x01;
    ReadWriteNRF(W, EN_AA, package, 1);

    // Lähetyksen epäonnistuessa uudelleenyritys 15 kertaa 750 µs väleillä
    package[0]=0x2F; // bitit 7:4 määräävät yritysten aikavälin
    ReadWriteNRF(W, SETUP_RETR, package, 1); // bitit 3:0 määräävät yritysten määrän

    // Valitaan datalinjojen määrä (data pipes)
    package[0]=0x01;
    ReadWriteNRF(W, EN_RXADDR, package, 1);
```

KUVA 14. Ensimmäinen osa nRF24L01+-lähetin-vastaanotinpiirin alustuksista

Piirin alustuksia suorittavan aliohjelman alussa on lyhyt 1,5 ms viive, jotta piiri ehtii varmasti suorittaa mahdolliset meneillään olevat toiminnallisuudet ennen alustuksia. Luotuun 5-tavuiseen taulukkoon tallennetaan SPI-väylälle lähetettävä varsinainen data, kuten esimerkiksi ohjelmoitavien rekisterien sisällöt. Ensimmäisenä alustetaan EN_AA-rekisterillä käyttöön automaattiset vastaukset piirien välillä, joka mahdollistaa automaattiset uudelleenlähetykset lähetyksen epäonnistuessa. Alustus tehdään datalinjaan 0, joka alustetaan käyttöön hieman myöhemmin eri komennolla. Toinen alustettava rekisteri SETUP_RETR määrää uudelleenyritysten lukumäärän ja väliajan. Tässä tapauksessa uudelleenyritysten määräksi asetettiin suurin mahdollinen, eli 15 yritystä. Yritykset uusitaan 750 µs välein, kunnes yritys onnistuu tai 15 uudelleenyrityskertaa tulee täyteen. Seuraavaksi valitaan EN_RXADDR-rekisteriin käytössä olevat datalinjat. Tässä

tapauksessa valitaan vain yksi datalinja (datalinja 0), koska langattomasti kommunikoivia laitteita on vain kaksi.

```
// RF-osoitteen tavumäärä 5 tavua
package[0]=0x03;
ReadWriteNRF(W, SETUP_AW, package, 1);

// RF-kanavan valinta taajuusalueella 2,400 GHz - 2,527 GHz, kanavaväli 1 MHz
package[0]=0x01;    // Kanava 1 taajuudella 2,401 GHz
ReadWriteNRF(W, RF_CH, package, 1);

// Siirtonopeus 250 kbps (min) ja lähetysteho 0dBm (max)
package[0]=0x26;  // 0b0010 0110 ;  bitit 5 ja 3 määräävät siirtonopeuden
ReadWriteNRF(W, RF_SETUP, package, 1); // bitit 2:1 määräävät lähetystehon

// RX_ADDR_P0 = TX_ADDR, kun EN_AA on käytössä!
// 5-tavuinen osoite vastaanottimelle
int i;
for(i=0; i<5; i++)
{
    package[i]=0x1B;
}
ReadWriteNRF(W, RX_ADDR_P0, package, 5);
```

KUVA 15. Toinen osa nRF24L01+-lähetin-vastaanotinpiirin alustuksista

SETUP_AW-rekisteriin määritetään langattomasti lähetettävän datapaketin osoitekentän pituus (kuva 15). Alustamalla kaksi vähiten merkitsevää bittiä rekisterissä 1-tilaan, osoitekentän kooksi tulee 5 tavua. Seuraavaksi alustetaan RF_CH-rekisteri, jonka seitsemän vähiten merkitsevää bittiä määrittelevät nRF24L01+-piirille käytössä olevan radiotaajuuskanavan. Lähettimelle ja vastaanottimelle on oleellista alustaa käyttöön sama kanava, koska vastaanottimen on kuunneltava lähettimen taajuutta. Arvolla 0x01 valituksi tulee kanava numero 1, jolloin lähetys tapahtuu 2,401 GHz:n taajuudella. Piirin kanavaväli on 1 MHz, joten seuraava kanava sijaitsee taajuudella 2,402 GHz. Suurimmalla siirtonopeudella (2 Mbit/s) lähetteen kaistanleveys on yli 1 MHz, jolloin aivan vierekkäisten kanavien käyttöä tulee välttää.

Kanavan valinnan jälkeen piirille määritetään RF_SETUP-rekisteriin käytettävä siirtonopeus ja lähetysteho. Kyseisen rekisterin bitit 5 ja 3 määräävät käytettävän siirtonopeuden, joka tässä työssä on 250 kbit/s. Bitit 2 ja 1 puolestaan määräävät tehon, jolla signaalin lähetys antennista tapahtuu. Lähetystehoksi alustetaan 0 dBm, jolloin siirtonopeudella 250 kbit/s piirin kuuluvuus on paras mahdollinen. Seuraavaksi RX_ADDR_P0-rekisteriin alustetaan 5-tavuinen osoite (0x1B1B1B1B1B) datalinjalta 0 vastanotettavalle paketille. Vastaanotto-osoitteen jälkeen alustetaan sama osoite lähetystilannetta varten TX_ADDR-rekisteriin (kuva 16).

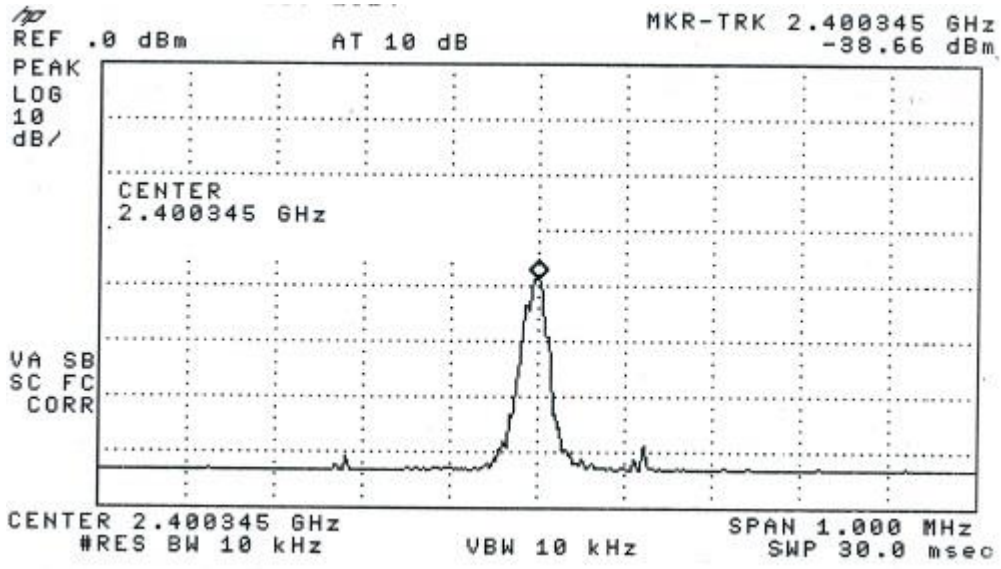
```
// 5-tavuinen osoite lähettimelle
for(i=0; i<5; i++)
{
    package[i]=0x1B;
}
ReadWriteNRF(W, TX_ADDR, package, 5);

// Lähetettävän datapaketin tavumäärä välillä 1-32 tavua
package[0]=bytes; // 3 tavua per lähetetty paketti (oltava sama lähettimessä ja vastaanottimessa)
ReadWriteNRF(W, RX_PW_P0, package, 1);

// Valitaan toimiiko piiri lähettimenä vai vastaanottimena ja asetetaan piiri power up -tilaan.
package[0]=0x1F; // Alustetaan aluksi vastaanottimeksi, 0x1E olisi lähetystila
ReadWriteNRF(W, CONFIG, package, 1);
receive = 1;

_delay_ms(1.5); // nRF24L01+ piirillä kestää hetken siirtyä standby-tilaan
}
```

KUVA 16. Viimeiset nRF24L01+-lähetin-vastaanotinpiirin alustukset

Rekistereihin RX_ADDR_P0 ja TX_ADDR tulee alustaa sama osoite (0x1B1B1B1B1B), kun EN_AA-rekisterissä on alustettu automaattiset kuittaukset käyttöön (nRF24L01+ datalehti, s. 60). Osoitetietojen jälkeen alustetaan lähetettävälle datapaketille varsinaista dataa varten varattava tavumäärä RX_PW_P0-rekisterillä. Suurin mahdollinen yhdessä paketissa radiotielle lähetettävä tavumäärä on 32 tavua. Tässä työssä tavumääräksi alustetaan vain kolme tavua, koska suuremmalle määrälle ei ole käyttöä. Tämänkin rekisterin tapauksessa on oleellista, että kaikille radiotiellä keskenään kommunikoiville piireille alustetaan sama tavumäärä käyttöön.

Viimeisenä alustetaan CONFIG-rekisteri, jossa asetetaan piiri joko lähetys- tai vastaanottotilaan. Molemmat piirit alustetaan käynnistyksen yhteydessä vastaanottotilaan, joten ne pitää erikseen alustaa lähetystilaan ennen lähetyksiä. Samalla piirille annetaan myös POWER UP -komento, jolloin piiri siirtyy lepotilasta valmiustilaan 1. Aliohjelman lopussa on vielä 1,5 ms viive, jolloin lähetin-vastaanotinpiirillä on varmasti riittävästi aikaa siirtyä valmiustilaan ennen ensimmäisiä lähetyksiä tai vastaanottoja. Tässä aliohjelmassa tehtyjen alustusten jälkeen piiri on täydessä valmiudessa vastaanottamaan lähetyksiä toiselta piiriltä.

Lähetin-vastaanotinpiirin RF_SETUP-rekisterissä on mahdollista asettaa jatkuva kantoaallon lähetys päälle alustamalla rekisterin seitsemäs bitti 1-tilaan. Ensimmäinen varsinainen merkki tiedonsiirron toimivuudesta saatiin asettamalla jatkuva kantoaallon lähetys ja mittaamalla lähetteen spektri spektrianalysaattorilla (kuva 17).



KUVA 17. Lähetin-vastaanotinpiirin kantoaallon spektri

Tällä tavalla varmistuttiin alustusten yhteydessä, että piirin rekisterien ohjelmointi todella onnistui. Kuvassa näkyvän spektrin kaistanleveys on hyvin kapea, koska piiri lähettää pelkkää kantoaaltoa. Lähetteen spektrin kaistanleveys leviää selvästi, kun kantoaaltoa moduloidaan digitaalisella hyötysignaalilla.

3.2.4 Keskeytyspalvelut

Ohjelmiston keskeytyspalvelut liittyvät kehitysalustan käyttöliittymän toteuttamiseen, sekä tiedonsiirtoon mikrokontrollerin ja nRF24L01+-lähetin-vastaanotinpiirin välillä. Aiemmin alustettu ajastinkeskeytyspalvelu TIMER1_COMPA_vect suoritetaan 10 ms:n välein, kun mikrokontrollerin sisäinen ajastin saavuttaa tietyn arvon (kuva 18).

```
/*************************************************************************************************/
ISR(TIMER1_COMPA_vect)  // Timer1-A-vertailukeskeytyspalvelu, suoritetaan 10 ms välein
{
    KesHetki = KesHetki + 20000;     // cmpA-keskeytyksen viritys 10 ms päähän
    OCR1A = KesHetki;
    KeskLippu = 1;       // Lipun asetus pääohjelmassa olevaa ehtolausetta varten
}
```

KUVA 18. 10 ms:n välein toteutettava ajastinkeskeytyspalvelu

Kyseisessä ajastinkeskeytysohjelmassa asetetaan aluksi seuraava keskeytys tapahtumaan uudestaan 10 ms:n kuluttua. Tämän jälkeen asetetaan keskeytyslippu arvoon 1 sen merkiksi, että keskeytys on tapahtunut. Tätä muuttujaa hyödynnetään pääohjelmasilmukassa kehitysalustan kytkintilojen lukemisessa. Kytkintilat siis luetaan pääohjelmasil-

mukassa 10 ms:n välein, jolloin säästyy huomattavasti prosessoriaikaa, kun kytkintiloja ei tarvitse lukea jokaisella pääohjelmasilmukan kierroksella. Ajastinkeskeytyksen lisäksi käytössä on nRF24L01+-lähetin-vastaanotinpiiriltä tulevan IRQ-signaalin laskureunalla käynnistyvä ulkoisen keskeytyksen palvelu INT6_vect (kuva 19).

```
/*****************************************************************************************/
ISR(INT6_vect)          // nRF24L01+ piirin aiheuttaman ulkoisen keskeytyksen palvelu
{
    PORTG &= ~0x01;   // CE low, piiri lopettaa kuuntelun
    if(receive == 1)  // Tämä toteutuu, jos piiri on alustettu vastaanottotilaan
    {
        //Datan luku mikrokontrollerille nRF24L01+ vastaanottorekisteristä
        data=ReadWriteNRF(R, R_RX_PAYLOAD, data, bytes);
        // Toiminnallisuus paketin sisällön perusteella
        if(data[0] == 0x93)
        {
            PORTA |= 0x40;      // Rele vetää -> LED syttyy
        }
        if(data[0] == 0x55)
        {
            PORTA &= ~0x40;     // Rele päästää -> LED sammuu
        }
    }
}
```

KUVA 19. INT6_vect-keskeytyspalveluohjelma

INT6_vect suoritetaan aina, kun nRF24L01+-lähetin-vastaanotinpiirin lähetys- tai vastaanottorekisteriin tulee dataa. Kyseisessä keskeytyspalvelussa ehtolauseella tutkitaan lipun perusteella, onko piiri lähetys- vai vastaanottotilassa. Ehtolauseen sisällä tulkitaan vastaanotetun paketin sisältämät tavut ja suoritetaan paketin datan perusteella toiminnallisuuksia. Ohjelmassa tavulla 0x93 kehitysalustan rele vetää, kun taas tavulla 0x55 kehitysalustan rele päästää.

3.2.5 Tiedonsiirto nRF24L01+-lähetin-vastaanotinpiirillä

Aliohjelmien ajo tapahtuu pääohjelmasilmukassa, jonka toistuva suorittaminen aloitetaan välittömästi kaikkien alustusten jälkeen (kuva 20).

```
/*****************************************************************************************/
int main(void)      // PÄÄOHJELMA
{
    IO_Init();              // IO-porttien alustus
    LCD_Init();             // LCD-moduulin alustus
    SPI_Init();             // SPI-väylän alustus
    Interrupt_Init();       // Keskeytysten alustus
    nRF24L01p_Init();       // nRF24L01+ -piirin alustus

    sei();      // Keskeytysten sallinta
    while(1)    // Pääohjelmasilmukka, toistetaan ikuisesti
    {
        if (KeskLippu == 1)     // Tämä ehto toteutuu 10 ms välein
        {
            KeskLippu = 0;
            Kytk = PINA;        // Kytkinten tilan luku muuttujaan
            Switches();
        }
        else
        {
            Receive_Payload();  // Mahdollisten saapuvien pakettien kuuntelua
            Reset();
        }
    }
    return 0;
}
```

KUVA 20. Pääohjelmasilmukka

Pääohjelmasilmukan alussa tutkitaan, onko keskeytyslippu asetettu ajastinkeskeytysohjelmassa kytkintilojen lukemisen merkiksi. Jos lippua ei ole asetettu 1-tilaan, siirrytään aliohjelmaan, jossa lähetin-vastaanotinpiiri asetetaan kuuntelemaan saapuvia paketteja (kuva 21).

```
/*****************************************************************************************/
// nRF24L01+ kuuntelee paketteja
void Receive_Payload(void)
{
    PORTG |= 0x01;      // CE high, piiri aloittaa kuuntelun
    _delay_ms(9);
    PORTG &= ~0x01;     // CE low, piiri lopettaa kuuntelun
}
```

KUVA 21. nRF24L01+ kuuntelee radiotieltä saapuvia paketteja

Receive_Payload-aliohjelmassa CE-signaali asetetaan 1-tilaan, jota seuraa 9 ms:n kestoinen viive. Viiveen jälkeen CE-signaali palautetaan 0-tilaan, jonka seurauksena lähetin-vastaanotinpiiri lopettaa pakettien vastaanottamisen. Jos 9 ms:n viiveen aikana piirille saapui paketti, niin IRQ-signaalilla on tapahtunut laskureuna, minkä seurauksena mikrokontrolleri ryhtyy suorittamaan keskeytyspalvelua INT6_vect, jossa pakettien sisältö tulkitaan (kuva 19). Tätä aliohjelmaa ajetaan pääohjelmasilmukassa aina, kun mikrokontrollerilla ei ole muuta tekemistä, eli 10 ms:n välein toteutettavaa kytkintilojen lukua. Pakettien kuuntelu on rajoitettu kerralla 9 ms:n mittaiseksi, jottei aliohjelman viive haittaa 10 ms:n välein toteutettavaa ajastinkeskeytysohjelmaa ja sitä kautta kytkintilojen lukua.

Pakettien lähetystä varten piiri täytyy aina alustaa lähetystilaan, koska kaksisuuntainen tiedonsiirto ei piirillä onnistu. Alustusta varten tehtiin Transmitter_Init-aliohjelma, joka ajetaan aina kun halutaan lähettää dataa (kuva 22).

```
/*************************************************************************************************/
// nRF24L01+ piirin alustus lähetystilaan
void Transmitter_Init(void)
{
    cli();      // Keskeytyskielto

    _delay_ms(1.5);
    uint8_t package[1];
    package[0]=0x1E;    // Paketille oikea sisältö piirin alustamiseksi lähetystilaan
    ReadWriteNRF(W, CONFIG, package, 1);    // Paketin lähetys piirille
    receive = 0;
    _delay_ms(1.5);     // Piirille aikaa siirtyä standby-tilaan
}
```

KUVA 22. nRF24L01+ -piirin alustus lähetystilaan

Aliohjelman ensimmäinen komento estää keskeytykset, jolloin lähetys ei pääse häiriintymään. Keskeytykset sallitaan taas lähetyksen jälkeen, kun piiri alustetaan takaisin vastaanottotilaan. Aliohjelmassa alustetaan nRF24L01+-piirin CONFIG-rekisterin nollas bitti 0-tilaan, jolloin piiri asettuu lähetystilaan. Aluksi pakettien lähetys toteutettiin täysin manuaalisesti, eli kehitysalustan kytkimiä painamalla. Kytkintoiminnot hoidetaan Switches-aliohjelmalla, joka suoritetaan pääohjelmasilmukassa 10 ms välein (kuva 23).

```
/*************************************************************************************************/
// Kytkintoimintoja suorittava aliohjelma
void Switches(void)
{
    // Ylin kytkin B1, rele vetää toisella kortilla
    if (((Kytk & 0x01) == 1) && (B1lask > 0))       B1lask--;
    else if (((Kytk & 0x01) == 0) && (B1lask > 0))    B1lask=9;
    else if (((Kytk & 0x01) == 0) && (B1lask == 0))
    {
        B1lask = 9;
        int i;
        for(i=0; i<3; i++)
        {
            dataout[i]=0x93;    // Lähetettävän paketin sisällöksi 0x93
        }
        Transmitter_Init();
        Transmit_Payload(dataout);
        Receiver_Init();
        Reset();
        receive = 1;
    }
```

KUVA 23. Kehitysalustan ylimmän kytkimen (B1) toiminnallisuudet

Kun kytkin tulkitaan aliohjelmassa painetuksi, kyseistä kytkintä vastaavat toiminnallisuudet alkavat. Kehitysalustan ylimmän kytkimen (B1) painallus kirjoittaa lähetin-vastaanotinpiirille lähetettävän 3-tavuisen taulukon jokaisen solun sisällöksi 0x93. Kyseinen tavu tulkitaan IRQ-signaalin aiheuttamassa keskeytyksessä komennoksi, jolla kehitysalustalla sijaitseva rele vetää (kuva 19). Tässä tilanteessa toki riittäisi 1-tavuinenkin paketti, tai vastaavasti tavun 0x93 kirjoitus vain yksittäiseen soluun lähetettävässä taulukossa. Selvyyden vuoksi radiotielle lähetettävä 3-tavuinen paketti kuitenkin

täytettiin samoilla tavuilla. Muut kytkimillä toteutetut toiminnallisuudet, kuten releen päästäminen alimmalla kytkimellä (B5), löytyvät ohjelmakoodista, joka on raportin liitteessä 3.

Kun lähetettävä paketti on valmis, ajetaan Transmitter_Init-aliohjelma, jolla nRF24L01+-lähetin-vastaanotinpiiri alustetaan lähetystilaan. Alustuksen valmistuttua ajetaan paketin lähettävä aliohjelma Transmit_Payload, joka saa parametrina radiotielle lähetettävän datan (kuva 24).

```
/*****************************************************************************************/
//nRF24L01+ paketin lähetysfunktio
void Transmit_Payload(uint8_t *T_data)
{
    // Vanhan datan poisto nRF24L01+ lähetysrekisteristä
    ReadWriteNRF(R, FLUSH_TX, T_data, 0);
    // muuttujassa olevan datan lähetys nRF24L01+ piirille
    ReadWriteNRF(R, W_TX_PAYLOAD, T_data, bytes);

    _delay_ms(1.5);
    PORTG |= 0x01;      // CE high, piiri aloittaa lähetyksen
    _delay_ms(2);
    PORTG &= ~0x01;     // CE low, piiri lopettaa lähetyksen
    _delay_ms(1.5);
}
```

KUVA 24. Aliohjelma, jolla nRF24L01+-piiri lähettää dataa radiotielle

Aliohjelmassa lähetetään aluksi lähetin-vastaanotinpiirille FLUSH_TX-komento, jolloin piiri tyhjentää lähetysrekisteristä siellä mahdollisesti vielä tallessa olevan vanhan datan. Poiston jälkeen piirille lähetetään komento W_TX_PAYLOAD, jolloin piiri tallentaa lähetysrekisteriin SPI-väylältä saapuvan paketin. Aliohjelmassa on lyhyt viive ennen CE-signaalin 1-tilaan kääntämistä, jotta paketti on varmasti ehtinyt lähetin-vastaanotinpiirin lähetysrekisteriin. Piirille annetaan viiveellä 2 ms aikaa lähettää lähetysrekisterissä oleva paketti antennista radiotielle, jonka jälkeen CE-signaali käännetään takaisin 0-tilaan ja lähetys loppuu. Lopussa on vielä pieni viive, joka antaa piirille aikaa siirtyä valmiustilaan. Lähetyksen loputtua kytkintoimintoja suorittavassa aliohjelmassa alustetaan piiri vielä takaisin vastaanottotilaan Receiver_Init-aliohjelmalla (kuva 25).

```
/*****************************************************************************************/
// nRF24L01+ piirin alustus vastaanottotilaan
void Receiver_Init(void)
{
    _delay_ms(1.5);
    uint8_t package[1];
    package[0]=0x1F;    // Paketille oikea sisältö piirin alustamiseksi vastaanottotilaan
    ReadWriteNRF(W, CONFIG, package, 1);// Paketin lähetys piirille
    _delay_ms(1.5);     // Piirille aikaa siirtyä standby-tilaan

    sei();      // Keskeytysten sallinta
}
```

KUVA 25. nRF24L01+ -piirin alustus vastaanottotilaan

Lähetyksen päätteeksi nRF24L01+-lähetin-vastaanotinpiirin keskeytysliput nollataan Reset-aliohjelmassa kirjoittamalla STATUS-rekisterin sisällöksi 0x70. Aliohjelmassa tyhjennetään myös lähetys- ja vastaanottorekisteri FLUSH-komennolla vanhasta datasta tulevia lähetyksiä ja vastaanottoja varten (kuva 26).

```
/*****************************************************************************************************/
// nRF24L01+ resetointi uutta vastaanottoa/lähetystä varten
void Reset(void)
{
    _delay_us(10);
    PORTB &= ~0x01;      // SS low
    _delay_us(10);
    SPI_MasterTransmit(W_REGISTER + STATUS); // Kirjoitus STATUS-rekisteriin
    _delay_us(10);
    SPI_MasterTransmit(0x70); // Keskeytyslippujen nollaus STATUS-rekisterissä
    _delay_us(10);
    PORTB |= 0x01;       // SS high
    // Vastaanottorekisterin tyhjennys vanhasta datasta
    ReadWriteNRF(R, FLUSH_RX, dataout, 0);
    // Vanhan datan poisto nRF24L01+ lähetysrekisteristä
    ReadWriteNRF(R, FLUSH_TX, dataout, 0);
}
```

KUVA 26. nRF24L01+ -piirin keskeytyslippujen nollaus

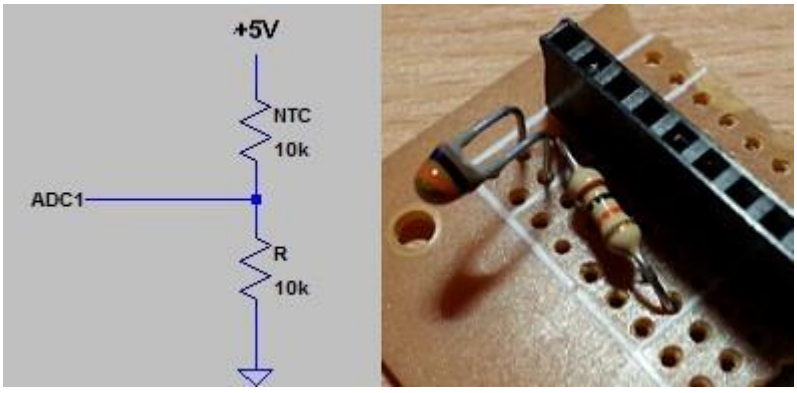
Reset-aliohjelman jälkeen piiri on taas valmis vastaanottamaan saapuvaa dataa tai suorittamaan uuden lähetyksen kytkimen painalluksesta. Tässä kappaleessa esitettyjen ohjelmakoodien avulla tiedonsiirto onnistuu kahden nRF24L01+-lähetin-vastaanotinpiirin välillä molempiin suuntiin.

4 ESIMERKKISOVELLUS

Tässä luvussa esitellään lämpötiladataa nRF24L01+-lähetin-vastaanotinpiirien välillä siirtävä esimerkkisovellus. Sovelluksen tavoitteena oli toteuttaa toiseen järjestelmän ohjainkorteista (myöhemmin anturikortti) lämpötila-anturi NTC-termistoria ja A/D-muunnosta hyödyntäen. Toisesta kortista tehtiin käyttöliittymäkortti, jonka kytkintoiminnoilla pystyy pyytämään anturikortilta lämpötilatiedon manuaalisesti tai asettamaan automaattisen päivityksen sekunnin välein. Mitattu lämpötilatieto tulostuu käyttöliittymäkortin näytölle yhden asteen resoluutiolla, koska esimerkkisovellusta varten toteutetun anturin tarkkuus ei riitä parempaan tulokseen.

4.1 Lämpötilan mittaaminen NTC-termistorilla [7], [8]

NTC-termistorin (myöhemmin NTC-vastus) käyttö lämpötilan mittaamisessa perustuu siihen, että sen vastusarvo muuttuu lämpötilan vaikutuksesta. Lyhenteenä NTC tarkoittaa, että kyseisellä vastuksella on negatiivinen lämpötilakerroin, eli NTC-vastuksen resistanssi pienenee lämpötilan kasvaessa. Lämpötila-anturin toteuttamisessa kytkettiin sarjaan normaali 10 kΩ vastus sekä NTC-vastus, jonka resistanssi 25 °C lämpötilassa on noin 10 kΩ (kuva 27).



KUVA 27. Lämpötila-anturin kytkentäkaavio ja käytännön toteutus

NTC-vastuksen ja normaalin vastuksen välillä tapahtuu jännitteenjako, joka mitataan ATMega128-mikrokontrollerin ADC1-kanavalla. Kun tiedetään NTC-vastuksen resistanssin olevan huoneenlämmössä 10 kΩ, tiedetään myös, että silloin jännitteenjaossa molempien vastusten yli vaikuttaa 2,5 V:n jännite. Lämpötilan kasvaessa NTC-

vastuksen yli jäävä jännite pienenee ja A/D-muunnoksella mitattava jännitearvo kasvaa. A/D-muunnos tehdään 8 bitin resoluutiolla. Kaavan 4.1 perusteella pienin havaittava muutos 5 V:n referenssijännitteellä on noin 19,5 mV (0x01).

$$\frac{5\ \mathrm{V}}{2^8} = 0{,}0195\ \mathrm{V} = 19{,}5\ \mathrm{mV} \qquad (4.1)$$

Kun tiedetään, että huoneenlämmössä jännitteenjaosta saatava muunnostulos on 0x80 ($128 \cdot 19{,}5\ \mathrm{mV} = 2{,}5\ \mathrm{V}$), voidaan A/D-muunnostuloksen perusteella laskea lämpötila. Lämpötilan laskemiseen käytettiin Steinhart-Hart -yhtälöä, jota käytetään yleisesti lämpötilan määrittämiseen termistoreilla.

4.2 Anturikortin ohjelmistoon tehdyt lisäykset [6], [7]

Ensimmäisenä anturikortin ohjelmistoon tehdään mikrokontrollerin A/D-muuntimen edellyttämät alustukset. A/D-muuntimen kanavat sijaitsevat mikrokontrollerin portissa F, joten kyseisen portin suuntarekisterissä alustetaan ADC1-kanavaa vastaava pinni inputiksi (kuva 28).

```
/*****************************************************************************************/
// AD-muuntimen alustus
void ADC_Init(void)
{
    DDRF = 0xFD;     // adc1/PF1 in muunnosta varten
    ADCSRA = 0x8F;   // ADEN=1 (ADC käyttöön), ADSC=1, ADIE=1 (ADC-keskeytyksen sallinta),
                     // Prescaler=0b111 jolloin näytteistys=xtal/128=125k
    ADMUX = 0x61;    // AVCC, ADLAR = 1, ADC1 käytössä, muut = 0
}
```

KUVA 28. Mikrokontrollerin A/D-muuntimen alustus

A/D-muuntimen asetukset alustetaan ADCSRA- ja ADMUX-rekistereissä (ATMega128 datalehti, s. 242–244). ADCSRA-rekisteriin alustetaan arvo 0x8F, jolloin A/D-muunnos sallitaan (ADEN=1), A/D-muunnoksen valmistuttua tapahtuva ADC_vect-keskeytys sallitaan (ADIE=1) ja näytteenottotaajuudeksi alustetaan esijakajalla 125 kHz (ADPS2=1, ADPS1=1 ja ADPS0=1).

ADMUX-rekisteriin alustetaan arvo 0x61, jolloin muunnoksessa käytettäväksi referenssijännitteeksi tulee mikrokontrollerin AREF-pinni (AVCC=1), muunnostuloksen 8 eniten merkitsevää bittiä tallentuvat ADCH-rekisteriin (ADLAR=1) (ATMega128 datalehti, s. 245) ja käytössä olevaksi A/D-muuntimen kanavaksi alustetaan ADC1 (MUX0=1).

Tarvittavien alustusten jälkeen IRQ-signaalin aiheuttamaan INT6_vect-keskeytyspalveluun lisättiin vertailulause A/D-muunnosta varten, jossa A/D-muunnos aloitetaan kun anturikortilla vastaanotetaan tavu 0xAD (kuva 29).

```
/*****************************************************************************************/
ISR(INT6_vect)      // nRF24L01+ piirin aiheuttaman ulkoisen keskeytyksen palvelu
{
    PORTG &= ~0x01;  // CE low, piiri lopettaa kuuntelun
    if(receive == 1) // Tämä toteutuu, jos piiri on alustettu vastaanottotilaan
    {
        //Datan luku mikrokontrollerille nRF24L01+ vastaanottorekisteristä
        data=ReadWriteNRF(R, R_RX_PAYLOAD, data, bytes);
        // Toiminnallisuus paketin sisällön perusteella
        if(data[0] == 0x93)
        {
            PORTA |= 0x40;          // Rele vetää -> LED syttyy
        }
        if(data[0] == 0x55)
        {
            PORTA &= ~0x40;         // Rele päästää -> LED sammuu
        }
        if(data[0] == 0xAD)
        {
            ADCSRA |= (1 << ADSC);  // AD-muunnos
            while(!(ADCSRA & (1<<ADSC)));
        }
    }
}
```

KUVA 29. A/D-muunnos IRQ-signaalin aiheuttamassa keskeytyspalvelussa

A/D-muunnos alkaa, kun ADCSRA-rekisterissä käännetään ADSC-lippu 1-tilaan. Tämän jälkeen odotetaan while-silmukassa, kunnes ADSC-lippu kääntyy takaisin 0-tilaan muunnoksen valmistumisen merkiksi. Koska ADC_vect-keskeytys on ADCSRA-rekisterissä sallittu, siirtyy mikrokontrolleri suorittamaan kyseistä keskeytyspalvelua välittömästi muunnoksen valmistuttua (kuva 30).

```
/*****************************************************************************************/
ISR (ADC_vect)      // AD-muunnoksen keskeytyspalvelu
{
    PORTA ^= 0x40;  // Kehitysalustan rele vaihtaa tilaa merkiksi
    adc=ADCH;       // Muunnostuloksen tallennus muuntimen datarekisteristä

    // Lämpötilan laskenta muunnostuloksesta
    calc=log(((2560000/adc) - 10000));
    calc=1/(0.001129148 + (0.000234125 * calc) + (0.0000000876741 * calc * calc * calc));
    temperature=calc-273.15; // Muunnos kelvineistä celsiusasteiksi

    int i;
    for(i=0; i<3; i++)
    {
        dataout[i]=0xAE;
        if(i==2)
        {
            // Lämpötilan tallennus lähetettävän taulukon soluun
            dataout[2]=temperature;
        }
    }
    SendTemp=1; // Merkkilippu 1-tilaan pääohjelmasilmukkaa varten
}
```

KUVA 30. A/D-muunnoksen valmistumisen jälkeen suoritettava keskeytysohjelma

ADC_vect-keskeytysohjelmassa muuntimen datarekisterissä odotteleva valmis muunnostulos tallennetaan muuttujaan. Työssä käytetyn NTC-vastuksen valmistaja ei ollut

tiedossa, joten täysin varmoja arvoja kuvan 30 yhtälöön ei pystytty sijoittamaan. Lämpötilan laskennassa hyödynnettiin internetistä löytyneitä ohjeita. Ohjeesta löytyneillä tyypillisillä arvoilla päästiin hyvin lähelle vallitsevaa lämpötilaa. Sijoittamalla laskutoimituksessa adc-muuttujan arvoksi 128, tulee lopputulokseksi pyöristysten jälkeen $25\ °C$. Laskutoimituksessa käytetty log-funktio laskee parametrina saamastaan lukuarvosta luonnollisen logaritmin. Log-funktio vaatii math-kirjaston sisällyttämisen ohjelmistoon toimiakseen (#include math.h).

Laskutoimitusten jälkeen tieto lämpötilasta lähetetään käyttöliittymäkortille ja alustetaan anturikortti takaisin vastaanottotilaan saapuvia paketteja varten. Tiedon lähetys tapahtuu vasta anturikortin pääohjelmasilmukassa, jolloin myös ajastinkeskeytys pääsee tarvittaessa suoritusvuoroon pitkän A/D-muunnoskeskeytyksen jälkeen (kuva 31).

```
Receive_Payload();  // Mahdollisten saapuvien pakettien kuuntelua
if(SendTemp==1)     // Jos vastaanotettiin lämpötilaa pyytävä paketti,
{                   // lähetetään lämpötila vastauksena
    SendTemp=0;              // Lipun nollaus
    Transmitter_Init();       // Piirin alustus lähettimeksi
    Transmit_Payload(dataout);  // Lämpötilan sisältävän paketin lähetys
    Receiver_Init();          // Piirin alustus vastaanottimeksi
    receive=1;
    LCDSendCommand(0x01);          // Näytön tyhjennys
    LCDSendTxt("   Temp sent   ");  // Tulostaa näytölle tekstin
}
Reset();
```

KUVA 31. Lämpötilan lähetys käyttöliittymäkortille

4.3 Käyttöliittymäkortin ohjelmistoon tehdyt lisäykset

Käyttöliittymäkortin lisäykset ovat huomattavasti vähäisemmät, mutta myös oleelliset. Aluksi kehitysalustan keskimmäiseen kytkimeen (B3) lisättiin toiminnallisuus, jolla pyydetään lämpötilatieto anturikortilta kerran asettamalla lähetetyn paketin sisällöksi 0xAD aiemmin esiteltyjen kytkintoimintojen perusteella. Käyttöliittymäkortille tehtiin myös sellainen ominaisuus, että lämpötilatiedon päivitys tapahtuu sekunnin välein (kuva 32).

```
// Oikea kytkin B4, lämpötilan automaattinen päivitys 1s välein päälle/pois
if (((Kytk & 0x08) == 0x08) && (B4lask > 0))      B4lask--;
else if (((Kytk & 0x08) == 0x00) && (B4lask > 0)) B4lask=9;
else if (((Kytk & 0x08) == 0x00) && (B4lask == 0))
{
    B4lask = 9;
    if(AutoUpdate==0)
    {
        AutoUpdate=1;
        Laskuri_1s=0;
    }
    else if(AutoUpdate==1)
    {
        AutoUpdate=0;
        Laskuri_1s=0;
    }

}
```

KUVA 32. Kytkin B4 asettaa automaattisen päivityksen päälle/pois

Kytkimellä B4 asetetaan lämpötilan päivitys joko päälle tai pois päältä. Varsinainen sekunnin välein tapahtuva lämpötilatiedon päivitys tapahtuu pääohjelmasilmukassa, jossa aikaa lasketaan 10 ms välein 1-tilaan asettuvan keskeytyslipun avulla (kuva 33).

```
while(1)    // Pääohjelmasilmukka, toistetaan ikuisesti
{
    if (KeskLippu == 1)         // Tämä ehto toteutuu 10 ms välein
    {
        KeskLippu = 0;
        Kytk = PINA;            // Kytkinten tilan luku muuttujaan
        Switches();
        if(AutoUpdate == 1)
        {
            Laskuri_1s++;           // 1s laskurin päivitys
            // Lämpötilan päivitys 1s välein, jos asetus on valittu
            if(Laskuri_1s == 100 && AutoUpdate == 1)
            {
                Laskuri_1s=0;
                int i;
                for(i=0; i<3; i++)
                {
                    dataout[i]=0xAD;        // Lähetettävän paketin sisällöksi 0xAD
                }
                Transmitter_Init();
                Transmit_Payload(dataout);
                Receiver_Init();
                Reset();
                receive = 1;
            }
        }
    }
    else
    {
        Receive_Payload();  // Mahdollisten saapuvien pakettien kuuntelua
        Reset();
    }
}
```

KUVA 33. Käyttöliittymäkortin pääohjelmasilmukka

Pääohjelmasilmukassa olevan laskurin arvoa kasvatetaan yhdellä 10 ms välein, jolloin yksi sekunti tulee täyteen sadan kierroksen jälkeen. Tämän jälkeen laskuri nollataan ja anturikortille lähetetään komento A/D-muunnoksen suorittamiseksi. Vastaus käyttöliittymäkortilla vastaanotetaan IRQ-signaalin laskureunan aiheuttamassa INT6_vect-keskeytyspalvelussa, aivan kuten muidenkin vastaanottojen tapauksessa (kuva 34).

```
/*****************************************************************************************************/
ISR(INT6_vect)          // nRF24L01+ piirin aiheuttaman ulkoisen keskeytyksen palvelu
{
    PORTG &= ~0x01;   // CE low, piiri lopettaa kuuntelun
    if(receive == 1)  // Tämä toteutuu, jos piiri on alustettu vastaanottotilaan
    {
        //Datan luku mikrokontrollerille nRF24L01+ vastaanottorekisteristä
        data=ReadWriteNRF(R, R_RX_PAYLOAD, data, bytes);
        // Toiminnallisuus paketin sisällön perusteella
        if(data[0] == 0x93)
        {
            PORTA |= 0x40;      // Rele vetää -> LED syttyy
        }
        if(data[0] == 0x55)
        {
            PORTA &= ~0x40;     // Rele päästää -> LED sammuu
        }
        if(data[0] == 0xAE)
        {
            LCDSendCommand(0x01);       // LCD -moduulin tyhjennys
            LCDSendInt(data[2]);        // Lämpötilan tulostus näytölle
        }
    }
}
```

KUVA 34. Vastaanotettu lämpötila tulostetaan kehitysalustan näytölle

Kun vastaanotettava paketti sisältää tavun 0xAE (anturikortilta saapuva lämpötilatieto), näytölle tulostetaan vastaanotetun taulukon soluun numero 2 tallennettu lämpötila. Tieto tulostuu näytölle vain asteen tarkkuudella, koska mittausjärjestelmä ei ole riittävän luotettava tarkemman lämpötilan mittaamiseksi.

5 YHTEENVETO

Tavoitteena oli toteuttaa sulautetussa järjestelmässä toimiva langaton tiedonsiirto kahdella nRF24L01+-lähetin-vastaanotinpiirillä. Tiedonsiirto tuli saada toimimaan mahdollisimman luotettavasti ja tehokkaasti lähetin-vastaanotinpiirien välillä. Lisäksi järjestelmällä toteutettiin esimerkkisovellus, joka lähettää lämpötilatietoa langattomasti järjestelmän sisällä. Ohjelmisto kehitettiin PC:llä, johon kytkettiin molemmat kehitysalustat. Kehitysalustojen samanaikainen ohjelmointi teki työskentelystä tehokasta, koska uusien ohjelmistoversioiden testaus oli nopeaa. Työn tavoitteisiin päästiin ja langaton tiedonsiirto saatiin toimimaan luotettavasti.

Työssä tehty ohjelmisto sisältää tarvittavat ohjelmarakenteet varsinaisen langattoman tiedonsiirron toteuttamiseen. Ohjelmiston pohjalta on helppoa lähteä työstämään langatonta tiedonsiirtoa moniin sovelluksiin, jos käytössä on vastaava lähetin-vastaanotinpiiri ja mikrokontrolleri. Ohjelmiston käyttökelpoisuuden kannalta oleellinen jatkokehitysmahdollisuus olisi tehdä nRF24L01+-lähetin-vastaanotinpiirille itsenäinen kirjasto, joka sisältää kaikki tarvittavat ohjelmarakenteet piirin ohjaamista varten. Itsenäisellä kirjastolla piirin käyttö olisi helppoa monissa projekteissa.

Työ oli yleisesti katsottuna tiedonsiirron ja ohjelmoinnin kannalta hyvinkin opettava, koska langaton tiedonsiirto sulautetuissa järjestelmissä oli entuudestaan tekijälle aiheena melko tuntematon. Aiemmat SPI-väylän käyttökokemukset olivat myös melko vähäiset, joten lähetin-vastaanotinpiirin ohjaaminen mikrokontrollerin SPI-väylän kautta oli hyödyllinen kokemus.

LÄHTEET

[1] Olimex AVR-MT-128-kehitysalustan tuotesivut. Luettu 21.3.2014.
https://www.olimex.com/Products/AVR/Development/AVR-MT128/

[2] Atmel AVR ATMega128 -datalehti.
http://www.atmel.fi/Images/doc2467_cn.pdf

[3] Nordic Semiconductor nRF24L01+ -datalehti.
http://www.nordicsemi.com/eng/Products/2.4GHz-RF/nRF24L01P

[4] Nordic Semiconductor nRF24L01 kirjasto, Stefan Engelke. Luettu 17.2.2014.
http://www.tinkerer.eu/AVRLib/nRF24L01

[5] Esimerkkiohjelmia, Kalle Löfgren. Luettu 3.3.2014.
http://gizmosnack.blogspot.fi/2013/04/tutorial-nrf24l01-and-avr.html

[6] Lämpötilan mittaus NTC-termistorilla. Luettu 21.3.2014.
http://www.instructables.com/id/Build-yourself-a-clock-and-thermometer/step3/Create-the-sketch-or-arduino/

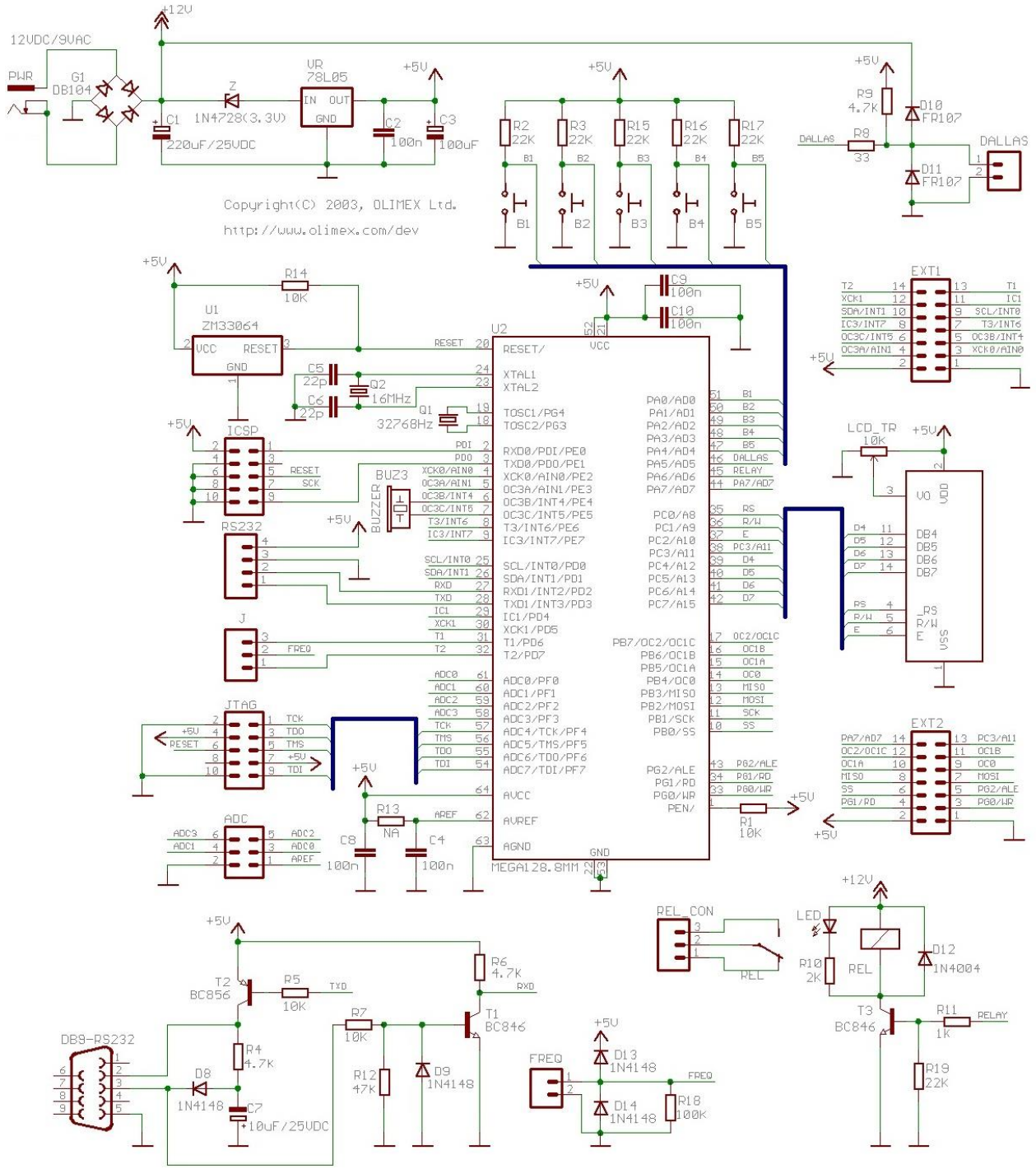
[7] Wikipedian artikkeli termistoreista. Luettu 21.3.2014.
http://en.wikipedia.org/wiki/Thermistor

[8] Wikipedian artikkeli Steinhart-Hart -yhtälöstä. Luettu 21.3.2014
http://en.wikipedia.org/wiki/Steinhart%E2%80%93Hart_equation

LIITTEET

Liite 1. AVR-MT-128-kehitysalustan kytkentäkaavio

https://www.olimex.com/Products/AVR/Development/AVR-MT128/



Liite 2. nRF24L01+-piirin rekisterit määrittelevä kirjasto (nRF24L01+.h)

http://www.tinkerer.eu/AVRLib/nRF24L01

```
// Piirin ohjelmoitavien rekisterien nimet ja osoitteet
#define CONFIG 0x00
#define EN_AA 0x01
#define EN_RXADDR 0x02
#define SETUP_AW 0x03
#define SETUP_RETR 0x04
#define RF_CH 0x05
#define RF_SETUP 0x06
#define STATUS 0x07
#define OBSERVE_TX 0x08
//#define CD 0x09                       // Tätä ei ole +-mallissa
#define RPD 0x09                        // +-Mallin vastaava rekisteri
#define RX_ADDR_P0 0x0A
#define RX_ADDR_P1 0x0B
#define RX_ADDR_P2 0x0C
#define RX_ADDR_P3 0x0D
#define RX_ADDR_P4 0x0E
#define RX_ADDR_P5 0x0F
#define TX_ADDR 0x10
#define RX_PW_P0 0x11
#define RX_PW_P1 0x12
#define RX_PW_P2 0x13
#define RX_PW_P3 0x14
#define RX_PW_P4 0x15
#define RX_PW_P5 0x16
#define FIFO_STATUS 0x17
#define DYNPD 0x1C
#define FEATURE 0x1D

// Rekisterien bitit numeroituna
#define MASK_RX_DR 6
#define MASK_TX_DS 5
#define MASK_MAX_RT 4
#define EN_CRC 3
#define CRCO 2
#define PWR_UP 1
#define PRIM_RX 0
#define ENAA_P5 5
#define ENAA_P4 4
#define ENAA_P3 3
#define ENAA_P2 2
#define ENAA_P1 1
#define ENAA_P0 0
#define ERX_P5 5
#define ERX_P4 4
#define ERX_P3 3
#define ERX_P2 2
#define ERX_P1 1
#define ERX_P0 0
#define AW 0
#define ARD 4
#define ARC 0
#define PLL_LOCK 4
//#define RF_DR 3               // Vanhassa mallissa vain 2 eri siirtonopeutta
#define RF_DR_LOW 5             //+-Mallissa kaksi bittiä määrää siirtonopeuden,
```

```
#define RF_DR_HIGH 3        // koska vaihtoehtoja on 3 (250 kbps, 1 Mbps ja 2 Mbps
//#define RF_PWR 6          // Vanhassa mallissa vain 2 eri lähetystehoa
#define RF_PWR_LOW 1        // + Mallissa kaksi bittiä määrää lähetystehon,
#define RF_PWR_HIGH 2       // koska vaihtoehtoja on 4 (0 dBm, -6 dBm, -12  dBm ja -18 dBm)
#define RX_DR 6
#define TX_DS 5
#define MAX_RT 4
#define RX_P_NO 1
#define TX_FULL 0
#define PLOS_CNT 4
#define ARC_CNT 0
#define TX_REUSE 6
#define FIFO_FULL 5
#define TX_EMPTY 4
#define RX_FULL 1
#define RX_EMPTY 0
#define DPL_P5 5
#define DPL_P4 4
#define DPL_P3 3
#define DPL_P2 2
#define DPL_P1 1
#define DPL_P0 0
#define EN_DPL 2
#define EN_ACK_PAY 1
#define EN_DYN_ACK 0

// SPI-väylää pitkin lähetettävät piirin komennot
#define R_REGISTER 0x00
#define W_REGISTER 0x20
#define R_RX_PAYLOAD 0x61
#define W_TX_PAYLOAD 0xA0
#define FLUSH_TX 0xE1
#define FLUSH_RX 0xE2
#define REUSE_TX_PL 0xE3
#define R_RX_PL_WID 0x60
#define W_ACK_PAYLOAD 0xA8
#define W_TX_PAYLOAD_NOACK 0xB0
#define REGISTER_MASK 0x1F
#define ACTIVATE 0x50
#define NOP 0xFF

// SPI-väylän lähetys- ja vastaanottofunktion lisäykset
#define W 1
#define R 0
```

Liite 3. Käyttöliittymäkortin ohjelmakoodi (kayttoliittyma.c)

```
// Olimex AVR-MT-128 (ATmega128) & Nordic Semiconductor nRF24L01+
// Opinnäytetyön ohjelmakoodi
// Ville Rosila
// 8.4.2014


/********************** KÄYTTÖLIITTYMÄKORTTI ********************************/
#include <avr/io.h>
#include <stdio.h>
#include <string.h>
#define F_CPU 16000000UL     // F_CPU -määritys delay-kirjastoa varten (16 MHz)
#include <util/delay.h>
#include <avr/interrupt.h>
#include "nRF24L01+.h"          // nRF24L01+ -piirin kirjasto

#define bytes 3  // Radiotiellä liikkuvien pakettien datan tavumäärä

// Aliohjelmien esittelyt
void IO_Init(void);
void SPI_Init(void);
void SPI_MasterTransmit(unsigned char);
char SPI_MasterTransmitRead(unsigned char);
void Interrupt_Init(void);
void Switches(void);
uint8_t *ReadWriteNRF(uint8_t ReadWrite, uint8_t NRFreg, uint8_t *package, uint8_t amount);
void nRF24L01p_Init(void);
void Transmitter_Init(void);
void Receiver_Init(void);
void Transmit_Payload(uint8_t *T_data);
void Receive_Payload(void);
void Reset(void);
void E_Pulse(void);
void LCD_Init(void);
void LCDSendChar(unsigned char a);
void LCDSendTxt(char* a);
void LCDSendCommand(unsigned char a);
void LCDSendInt_Old(int a);
void LCDSendInt(long a);

// Tarvittavat muuttujat
unsigned char
        lcd_data,
        *data,
        *datain[bytes],
        dataout[bytes];
volatile unsigned char
        receive,
        Kytk,
        KeskLippu = 0,
        B1lask,
        B2lask,
        B3lask,
        B4lask,
        B5lask,
        AutoUpdate = 0;
volatile unsigned int
        KesHetki,
        Laskuri_1s=0;
```

```
/*****************************************************************************/
ISR(TIMER1_COMPA_vect)    // Timer1-A-vertailukeskeytyspalvelu, suoritetaan 10 ms välein
{
        KesHetki = KesHetki + 20000; // cmpA-keskeytyksen viritys 10 ms päähän
        OCR1A = KesHetki;
        KeskLippu = 1;              // Lipun asetus pääohjelmassa olevaa ehtolausetta varten
}


/*****************************************************************************/
ISR(INT6_vect) // nRF24L01+ piirin aiheuttaman ulkoisen keskeytyksen palvelu
{
        PORTG &= ~0x01; // CE low, piiri lopettaa kuuntelun
        if(receive == 1) // Tämä toteutuu, jos piiri on alustettu vastaanottotilaan
        {
                //Datan luku mikrokontrollerille nRF24L01+ vastaanottorekisteristä
                data=ReadWriteNRF(R, R_RX_PAYLOAD, data, bytes);
                // Toiminnallisuus paketin sisällön perusteella
                if(data[0] == 0x93)
                {
                        PORTA |= 0x40;              // Rele vetää -> LED syttyy
                }
                if(data[0] == 0x55)
                {
                        PORTA &= ~0x40;             // Rele päästää -> LED sammuu
                }
                if(data[0] == 0xAE)
                {
                        LCDSendCommand(0x01); // LCD -moduulin tyhjennys
                        LCDSendInt(data[2]); // Lämpötilan tulostus näytölle
                }
        }
}


/*****************************************************************************/
int main(void)   // PÄÄOHJELMA
{
        IO_Init();                  // IO-porttien alustus
        LCD_Init();                 // LCD-moduulin alustus
        SPI_Init();                 // SPI-väylän alustus
        Interrupt_Init();           // Keskeytysten alustus
        nRF24L01p_Init();           // nRF24L01+ -piirin alustus

        sei();      // Keskeytysten sallinta
        while(1) // Pääohjelmasilmukka, toistetaan ikuisesti
        {
                if (KeskLippu == 1) // Tämä ehto toteutuu 10 ms välein
                {
                        KeskLippu = 0;
                        Kytk = PINA; // Kytkinten tilan luku muuttujaan
                        Switches();
                        if(AutoUpdate == 1)
                        {
                                Laskuri_1s++;       // 1s laskurin päivitys
                                // Lämpötilan päivitys 1s välein, jos asetus on valittu
                                if(Laskuri_1s == 100 && AutoUpdate == 1)
                                {
                                        Laskuri_1s=0;
```

```
                                int i;
                                for(i=0; i<3; i++)
                                {
                                        dataout[i]=0xAD; // Lähetettävän paketin sisällöksi 0xAD
                                }
                                Transmitter_Init();
                                Transmit_Payload(dataout);
                                Receiver_Init();
                                Reset();
                                receive = 1;
                        }
                }
        }
        else
        {
                Receive_Payload(); // Mahdollisten saapuvien pakettien kuuntelua
                Reset();
        }
    }
    return 0;
}


/*****************************************************************************************/
// Kytkintoimintoja suorittava aliohjelma
void Switches(void)
{
    // Ylin kytkin B1, rele vetää toisella kortilla
    if (((Kytk & 0x01) == 1) && (B1lask > 0)) B1lask--;
    else if (((Kytk & 0x01) == 0) && (B1lask > 0)) B1lask=9;
    else if (((Kytk & 0x01) == 0) && (B1lask == 0))
    {
        B1lask = 9;
        int i;
        for(i=0; i<3; i++)
        {
            dataout[i]=0x93;            // Lähetettävän paketin sisällöksi 0x93
        }
        Transmitter_Init();
        Transmit_Payload(dataout);
        Receiver_Init();
        Reset();
        receive = 1;
    }


    // Vasen kytkin B2, näyttötesti
    if (((Kytk & 0x02) == 0x02) && (B2lask > 0))    B2lask--;
    else if (((Kytk & 0x02) == 0x00) && (B2lask > 0)) B2lask=9;
    else if (((Kytk&0x02) == 0x00) && (B2lask == 0))
    {
        B2lask = 9;
        LCDSendCommand(0x01);  //Clear Display
        LCDSendTxt("    Test    ");          //Tulostaa näytölle testiksi
    }


    // Keskimmäinen kytkin B3, lämpötilan päivitys kerran
    if (((Kytk & 0x04) == 0x04) && (B3lask > 0))    B3lask--;
    else if (((Kytk & 0x04) == 0x00) && (B3lask > 0)) B3lask=9;
    else if (((Kytk & 0x04) == 0x00) && (B3lask == 0))
```

```
	{
		B3lask = 9;
		int i;
		for(i=0; i<3; i++)
		{
			dataout[i]=0xAD;		// Lähetettävän paketin sisällöksi 0xAD
		}
		Transmitter_Init();
		Transmit_Payload(dataout);
		Receiver_Init();
		Reset();
		receive = 1;
	}

	// Oikea kytkin B4, lämpötilan automaattinen päivitys 1s välein päälle/pois
	if (((Kytk & 0x08) == 0x08) && (B4lask > 0))	B4lask--;
	else if (((Kytk & 0x08) == 0x00) && (B4lask > 0)) B4lask=9;
	else if (((Kytk & 0x08) == 0x00) && (B4lask == 0))
	{
		B4lask = 9;
		if(AutoUpdate==0)
		{
			AutoUpdate=1;
			Laskuri_1s=0;
		}
		else if(AutoUpdate==1)
		{
			AutoUpdate=0;
			Laskuri_1s=0;
		}
	}

	// Alin kytkin B5, rele päästää toisella kortilla
	if (((Kytk & 0x10) == 0x10) && (B5lask > 0))	B5lask--;
	else if (((Kytk & 0x10) == 0x00) && (B5lask > 0)) B5lask=9;
	else if (((Kytk & 0x10) == 0x00) && (B5lask==0))
	{
		B5lask = 9;
		int i;
		for(i=0; i<3; i++)
		{
			dataout[i]=0x55;  // Lähetettävän paketin sisällöksi 0x55
		}
		Transmitter_Init();
		Transmit_Payload(dataout);
		Receiver_Init();
		Reset();
		receive = 1;
	}
}

/*****************************************************************************/
// IO-porttien alustukset
void IO_Init(void)
{
	DDRA = 0x40;  // PA1...PA4/Kytkimet = in, PA6/Rele = out
	DDRG = 0x01;  // PG0/CE = out
}

/*****************************************************************************/
// SPI-väylän alustukset
```

```
void SPI_Init(void)
{
	DDRB = 0x07;		// PB0/SS = out, PB1/CLK = out, PB2/MOSI = out, PB3/MISO = in
	SPCR = 0x52;		// SPI Enable=1, Master=1, SPR1=1 ja SPR0=0, jolloin
				// fosc/64 = 250kHz eli 250 kbit/s siirtonopeus
	PORTB |= 0x01;		// SS high alustus
	PORTG &= ~0x01;		// CE low alustus
}

/*****************************************************************************/
// Yksittäisen tavun lähetys SPI-väylällä
void SPI_MasterTransmit(unsigned char cData)
{
	SPDR = cData;			// Tavu SPI-väylän datarekisteriin
	while(!(SPSR & (1<<SPIF))); // Odotetaan, että lähetys loppuu
}

/*****************************************************************************/
// Yksittäisen tavun lähetys ja luku SPI-väylällä
char SPI_MasterTransmitRead(unsigned char cData)
{
	SPDR = cData;				// Tavu SPI-väylän datarekisteriin
	while(!(SPSR & (1<<SPIF)));		// Odotetaan, että lähetys loppuu
	return SPDR;				// Paluuarvona vastaanotettu data
}

/*****************************************************************************/
// Keskeytysten alustus
void Interrupt_Init(void)
{
	DDRE &= ~0x40;		// PE6/INT6 = in (IRQ)
	EICRB = 0x20;		// INT6 laskeva reuna aiheuttaa keskeytyspyynnön (bitit 5:4 = 10)
	EIMSK = 0x40;		// INT6 -keskeytyksen sallinta

	KesHetki = 20000;	// Ensimmäisen OCR1A-keskeytyksen ajanhetki 10 ms resetistä
	OCR1A = KesHetki;
	TCCR1B = 0x02;		// 0000 0010 nonPWM, clk/8
	TIMSK = 0x10;		// 0001 0000
}

/*****************************************************************************/
// Kommunikaatio nRF24L01+ -piirin kanssa SPI-väylän avulla
uint8_t *ReadWriteNRF(uint8_t ReadWrite, uint8_t NRFreg, uint8_t *package, uint8_t amount)
{
	static uint8_t ret[bytes];

	// Kirjoituskomento
	if(ReadWrite == W)
	{
		NRFreg = W_REGISTER + NRFreg;
	}

	_delay_us(10);
	PORTB &= ~0x01;		// SS low
	_delay_us(10);
	SPI_MasterTransmit(NRFreg);		// Luku/kirjoituskomennon lähetys piirille
	_delay_us(10);

	int i;
	for(i=0; i<amount; i++)
	{
```

```
		if(ReadWrite == R && NRFreg != W_TX_PAYLOAD)		// Datan luku piiriltä
		{
			ret[i]=SPI_MasterTransmitRead(NOP);
			_delay_us(10);
		}
		else			// Datan lähetys piirille
		{
		SPI_MasterTransmit(package[i]);
		_delay_us(10);
		}
	}
	PORTB |= 0x01;		// SS high
	return ret;			// Paluuarvona piiriltä luettu paketti
}


/*****************************************************************************/
// nRF24L01+ alustus
void nRF24L01p_Init(void)
{
	_delay_ms(1.5);		// Pieni delay, jotta piiri on varmasti standby-tilassa

	uint8_t package[5];

	// Sallitaan automaattiset vastaukset vastaanottavalta piiriltä (auto-acknowledgements)
	// Tämä mahdollistaa automaattiset lähetysten uudelleenyritykset
	package[0]=0x01;
	ReadWriteNRF(W, EN_AA, package, 1);

	// Lähetyksen epäonnistuessa uudelleenyritys 15 kertaa 750 µs väleillä
	package[0]=0x2F; // bitit 7:4 määräävät yritysten aikavälin ja bitit 3:0 määräävät yritysten määrän
	ReadWriteNRF(W, SETUP_RETR, package, 1);

	// Valitaan datalinjojen määrä (data pipes)
	package[0]=0x01;
	ReadWriteNRF(W, EN_RXADDR, package, 1);

	// RF-osoitteen tavumäärä 5 tavua
	package[0]=0x03;
	ReadWriteNRF(W, SETUP_AW, package, 1);

	// RF-kanavan valinta taajuusalueella 2,400 GHz - 2,527 GHz, kanavaväli 1 MHz
	package[0]=0x01;		// Kanava 1 taajuudella 2,401 GHz
	ReadWriteNRF(W, RF_CH, package, 1);

	// Siirtonopeus 250 kbps (min) ja lähetysteho 0dBm (max)
	package[0]=0x26;		// 0b0010 0110 ;  bitit 5 ja 3 määräävät siirtonopeuden
				//bitit 2:1 määräävät lähetystehon
	ReadWriteNRF(W, RF_SETUP, package, 1);

	// RX_ADDR_P0 = TX_ADDR, kun EN_AA on käytössä!
	// 5-tavuinen osoite vastaanottimelle
	int i;
	for(i=0; i<5; i++)
	{
		package[i]=0x1B;
	}
	ReadWriteNRF(W, RX_ADDR_P0, package, 5);

	// 5-tavuinen osoite lähettimelle
	for(i=0; i<5; i++)
```

```
	{
		package[i]=0x1B;
	}
	ReadWriteNRF(W, TX_ADDR, package, 5);

	// Lähetettävän datapaketin tavumäärä välillä 1-32 tavua
	package[0]=bytes; // 3 tavua per lähetetty paketti (oltava sama lähettimessä ja vastaanottimessa)
	ReadWriteNRF(W, RX_PW_P0, package, 1);


	// Valitaan toimiiko piiri lähettimenä vai vastaanottimena ja asetetaan piiri power up -tilaan.
	package[0]=0x1F; // Alustetaan aluksi vastaanottimeksi, 0x1E olisi lähetystila
	ReadWriteNRF(W, CONFIG, package, 1);
	receive = 1;

	_delay_ms(1.5); // nRF24L01+ piirillä kestää 1,5 ms siirtyä standby-tilaan
}

/*****************************************************************************************/
// nRF24L01+ piirin alustus lähetystilaan
void Transmitter_Init(void)
{
	cli();	// Keskeytyskielto

	_delay_ms(1.5);
	uint8_t package[1];
	package[0]=0x1E;		// Paketille oikea sisältö piirin alustamiseksi lähetystilaan
	ReadWriteNRF(W, CONFIG, package, 1); // Paketin lähetys piirille
	receive = 0;
	_delay_ms(1.5);			// Piirille aikaa siirtyä standby-tilaan
}

/*****************************************************************************************/
// nRF24L01+ piirin alustus vastaanottotilaan
void Receiver_Init(void)
{
	_delay_ms(1.5);
	uint8_t package[1];
	package[0]=0x1F;		// Paketille oikea sisältö piirin alustamiseksi vastaanottotilaan
	ReadWriteNRF(W, CONFIG, package, 1); // Paketin lähetys piirille
	_delay_ms(1.5);				// Piirille aikaa siirtyä standby-tilaan

	sei();	// Keskeytysten sallinta
}


/*****************************************************************************************/
//nRF24L01+ paketin lähetysfunktio
void Transmit_Payload(uint8_t *T_data)
{
	// Vanhan datan poisto nRF24L01+ lähetysrekisteristä
	ReadWriteNRF(R, FLUSH_TX, T_data, 0);
	// muuttujassa olevan datan lähetys nRF24L01+ piirille
	ReadWriteNRF(R, W_TX_PAYLOAD, T_data, bytes);

	_delay_ms(1.5);
	PORTG |= 0x01;		// CE high, piiri aloittaa lähetyksen
	_delay_ms(2);
	PORTG &= ~0x01;		// CE low, piiri lopettaa lähetyksen
	_delay_ms(1.5);
}
```

```
/*****************************************************************************/
// nRF24L01+ kuuntelee paketteja
void Receive_Payload(void)
{
	PORTG |= 0x01;		// CE high, piiri aloittaa kuuntelun
	_delay_ms(9);
	PORTG &= ~0x01;		// CE low, piiri lopettaa kuuntelun
}

/*****************************************************************************/
// nRF24L01+ resetointi uutta vastaanottoa/lähetystä varten
void Reset(void)
{
	_delay_us(10);
	PORTB &= ~0x01;		// SS low
	_delay_us(10);
	SPI_MasterTransmit(W_REGISTER + STATUS); // Kirjoitus STATUS-rekisteriin
	_delay_us(10);
	SPI_MasterTransmit(0x70);		// Keskeytyslippujen nollaus STATUS-rekisterissä
	_delay_us(10);
	PORTB |= 0x01;		// SS high
	// Vastaanottorekisterin tyhjennys vanhasta datasta
	ReadWriteNRF(R, FLUSH_RX, dataout, 0);
	// Vanhan datan poisto nRF24L01+ lähetysrekisteristä
	ReadWriteNRF(R, FLUSH_TX, dataout, 0);
}

/*****************************************************************************/
//  LCD -moduulin ohjaukset, lainattu Olimexin sivuilta
void E_Pulse()
{
	PORTC |= 0x04;
	_delay_ms(2);
	 PORTC &= ~0x04;
}

void LCD_Init()
{
	DDRC = 0xF7;		/* LCD: D7 D6 D5 D4 nc E RW RS */
	PORTC = PORTC & 0b11111110;

	_delay_ms(50);

	PORTC = 0b00110000;	//set D4, D5 port to 1
	E_Pulse();		//high->low to E port (pulse)
	_delay_ms(6);
	PORTC = 0b00110000;	//set D4, D5 port to 1
	E_Pulse();		//high->low to E port (pulse)
	_delay_ms(6);
	PORTC = 0b00110000;	//set D4, D5 port to 1
	E_Pulse();		//high->low to E port (pulse)
	_delay_ms(6);
	PORTC = 0b00100000;	//set D4 to 0, D5 port to 1
	E_Pulse();		//high->low to E port (pulse)

	LCDSendCommand(0x0000000C); //Turn ON Display
}

void LCDSendChar(unsigned char a)
{
	lcd_data = 0b00001111 | a;		//get high 4 bits
```

```
	PORTC = (PORTC | 0b11110000) & lcd_data; //set D4-D7
	PORTC = PORTC | 0b00000001;		//set RS port to 1
	E_Pulse();				//pulse to set D4-D7 bits

	lcd_data = a<<4;			//get low 4 bits
	PORTC = (PORTC & 0b00001111) | lcd_data; //clear D4-D7
	PORTC = PORTC | 0b00000001;		//set RS port to 1 -> display set to command mode
	E_Pulse();				//pulse to set d4-d7 bits
}

void LCDSendTxt(char* a)
{
	int Temp;
	for(Temp=0; Temp<strlen(a); Temp++)
	{
		LCDSendChar(a[Temp]);
	}
}

void LCDSendCommand(unsigned char a)
{
	lcd_data = 0b00001111 | a;		//get high 4 bits
	PORTC = (PORTC | 0b11110000) & lcd_data; //set D4-D7
	PORTC = PORTC & 0b11111110;		//set RS port to 0
	E_Pulse();				//pulse to set D4-D7 bits

	lcd_data = a<<4;			//get low 4 bits
	PORTC = (PORTC & 0b00001111) | lcd_data; //set D4-D7
	PORTC = PORTC & 0b11111110;		//set RS port to 0 -> display set to command mode
	E_Pulse();				//pulse to set d4-d7 bits
}

void LCDSendInt(long a)
{
	int C[20];
	unsigned char Temp=0, NumLen = 0;
	if (a < 0)
	{
		LCDSendChar('-');
		a = -a;
	}
	do
	{
		Temp++;
		C[Temp] = a % 10;
		a = a/10;
	}
	while (a);
	NumLen = Temp;
	for (Temp = NumLen; Temp>0; Temp--) LCDSendChar(C[Temp] + 48);
}
```

Liite 4. Anturikortin ohjelmistoon tarvittavat lisäykset

```
#include <math.h>

void ADC_Init(void);

volatile unsigned char SendTemp;
unsigned char adc;
float calc;
float temperature;


/*************************************************************************/
// AD-muuntimen alustus
void ADC_Init(void)
{
	DDRF = 0xFD;		// adc1/PF1 in muunnosta varten
	ADCSRA = 0x8F;		// ADEN=1 (ADC käyttöön), ADSC=1,
				// ADIE=1 (ADC-keskeytyksen sallinta),
				// Prescaler=0b111 jolloin näytteistys=xtal/128=125k
	ADMUX = 0x61;		// AVCC, ADLAR = 1, ADC1 käytössä, muut = 0
}


/*************************************************************************/
ISR(INT6_vect) // nRF24L01+ piirin aiheuttaman ulkoisen keskeytyksen palvelu
{
	PORTG &= ~0x01;		// CE low, piiri lopettaa kuuntelun
	if(receive == 1)	// Tämä toteutuu, jos piiri on alustettu vastaanottotilaan
	{
		//Datan luku mikrokontrollerille nRF24L01+ vastaanottorekisteristä
		data=ReadWriteNRF(R, R_RX_PAYLOAD, data, bytes);
		// Toiminnallisuus paketin sisällön perusteella
		if(data[0] == 0x93)
		{
			PORTA |= 0x40;  // Rele vetää -> LED syttyy
		}
		if(data[0] == 0x55)
		{
			PORTA &= ~0x40;  // Rele päästää -> LED sammuu
		}
		if(data[0] == 0xAD)
		{
			ADCSRA |= (1 << ADSC);  // AD-muunnos
			while(!(ADCSRA & (1<<ADSC)));
		}
	}
}


/*************************************************************************/
ISR (ADC_vect) // AD-muunnoksen keskeytyspalvelu
{
	PORTA ^= 0x40;		// Kehitysalustan rele vaihtaa tilaa muunnoksen valmistumisen merkiksi
	adc=ADCH;		// Muunnostuloksen tallennus muuntimen datarekisteristä

	// Lämpötilan laskenta muunnostuloksesta
	calc=log(((2560000/adc) - 10000));
	calc=1/(0.001129148 + (0.000234125 * calc) + (0.0000000876741 * calc * calc * calc));
	temperature=calc-273.15; // Muunnos kelvineistä celsiusasteiksi
```

```
        int i;
        for(i=0; i<3; i++)
        {
                dataout[i]=0xAE;
                if(i==2)
                {
                        // Lämpötilan tallennus lähetettävän taulukon soluun
                        dataout[2]=temperature;
                }
        }
        SendTemp=1;             // Merkkilippu 1-tilaan pääohjelmasilmukkaa varten
}

/*****************************************************************************/
ADC_Init();     // AD-muuntimen alustus
while(1)        // Pääohjelmasilmukka
{
        if (KeskLippu == 1)     // Toteutuu 10 ms välein, timer1_compa-kesk. asettaa
        {
                KeskLippu = 0;
                Kytk = PINA;    // Kytkinten tilan luku muuttujaan
                Switches();
         }
        else
        {
                Receive_Payload();              // Mahdollisten saapuvien pakettien kuuntelua
                if(SendTemp==1)                 // Jos vastaanotettiin lämpötilaa pyytävä paketti,
                {                               // lähetetään lämpötila vastauksena
                        SendTemp=0;                             // Lipun nollaus
                        Transmitter_Init();                     // Piirin alustus lähettimeksi
                        Transmit_Payload(dataout);              // Lämpötilan sisältävän paketin lähetys
                        Receiver_Init();                        // Piirin alustus vastaanottimeksi
                        receive=1;
                        LCDSendCommand(0x01);                   // Näytön tyhjennys
                        LCDSendTxt("   Temp sent   ");          // Tulostaa näytölle tekstin
                }
                Reset();
        }
}
```